ELISA NASCIMENTO

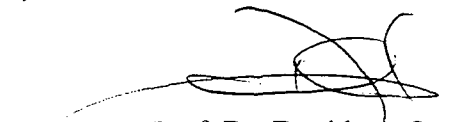
REPRESENTAÇÃO E FERRAMENTAS PARA APOIAR A PROGRAMAÇÃO DE DISPOSITIVOS DIGITAIS COM MECANISMOS DE INTERPRETAÇÃO INDEPENDENTES DE LINGUAGEM

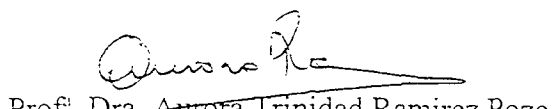
Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Curso de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Alexandre I. Direne

CURITIBA

2000

Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Comissão Examinadora da defesa de Dissertação de Mestrado em Informática, da aluna Elisa Nascimento, avaliamos o trabalho **"REPRESENTAÇÃO E FERRAMENTAS PARA APOIAR A PROGRAMAÇÃO DE DISPOSITIVOS DIGITAIS COM MECANISMOS DE INTERPRETAÇÃO INDEPENDENTES DE LINGUAGEM"**, cuja defesa foi realizada no dia 31 de março de 2000. Após a Avaliação, decidimos pela Aprovação da Candidata.

Curitiba, 31 de março de 2000.

Prof. Dr. Alexandre Ibrahim Direne
Presidente Dinf / UFPR

Prof. Dr. Davidson Cury
M / E. UFES

Prof[a]. Dra. Aurora Trinidad Ramirez Pozo
Dinf / UFPR

AGRADECIMENTOS

Ao meu orientador Alexandre pela ajuda prestada durante este *longo* período. Mesmo quando nossas agendas não estavam em sintonia, foi possível continuar o trabalho... principalmente nesta reta final. Ao meu namorado Dieferson pelo incentivo quando era atacada por neuroses se devia continuar ou não com o trabalho; e que soube entender os finais de semana dedicados à escrita, sendo este o único momento em que podemos ficar juntos. Aos meus pais, que durante todo o tempo não efetuaram cobrança, apenas incentivaram o término "o mais rápido possível". A todos os colegas de turma e àqueles que, de uma maneira direta ou indireta, contribuíram para este trabalho (Aldri, Andrey, Claudio, Edgar, Felipe, Rodrigo e tantos outros). Aos professores e funcionários do Departamento de Informática da Universidade Federal do Paraná, com os quais convivi , aprendi (e discuti) durante dois anos.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

APPLE - Árvore Para Prescrever a Lógica da solução de Exercícios

CDCT - Central Digital de Comutação Telefônica

PROTTEL-D - PROjeto de sistemas Tutoriais para TELefonia Digital

MAEL - Modelo de Apoio a Erros de Lógica

MAES - Modelo de Apoio a Erros de aspecto Sintático

MEDAL - MEta-linguagem para a Descrição de Aspectos da Lógica de solução de um problema

MELON - MEta-Language for defining OperatioN commands

SATELIT - Sistema de Apoio à TELefonia com Inteligência integrada ao Treinamento

SPX2000 - Central Digital de Comutação de pequeno porte

EWSD - Central Digital de Comutação de grande porte

RESUMO

Este trabalho apresenta o projeto de ferramentas para apoiar o treinamento de novatos em operação e manutenção de centrais digitais de comutação telefônica. Estas ferramentas fazem parte de uma *shell* já implementada que contém um simulador. O simulador oferece reação inteligente aos erros de sintaxe cometidos pelo operador iniciante através de uma ferramenta descrita neste trabalho. Futuramente, o sistema vai oferecer também apoio à lógica de programação da central através da outra ferramenta proposta neste trabalho. Entre os objetivos a serem alcançados pelo sistema tutorial no qual estas ferramentas trabalham (denominado SATELIT) estão: diminuir o tempo necessário de treinamento e aumentar o potencial de diagnóstico de erros pelo operador.

ABSTRACT

This work presents the design of tools to support training for novice operators of digital telephony stations. These tools are part of an implemented computer-based *shell* that embodies a simulator. The simulator offers intelligent feedback to syntactic error entered by novice operators using one tool described in this work. In the future, the system will also give support to logic errors using the other tool proposed in this work. Among the goals of SATELIT system, in which these tools are inserted, are: decrease the necessary time designated to novice training and increase the potential of error detection by operators.

1 INTRODUÇÃO

Este trabalho apresenta o projeto de uma parte de uma "*shell*" de sistema tutorial inteligente utilizado para apoiar o treinamento de operação e manutenção de Centrais Digitais de Comutação Telefônica (CDCT). Uma CDCT é operada através de uma linguagem formal de programação. Portanto, este trabalho tem em foco alguns dos problemas no ensino de linguagens formais. Um aluno de programação somente pode ser considerado apto para resolver problemas algorítmicos de maneira satisfatória quando adquire diversas habilidades que podem ser divididas em princípio e perícia (du BOULAY; SOTHCOTT, 1987). O princípio de programação diz respeito à sintaxe (como deve ser escrito) e à semântica (o que faz) isolada de cada comando. Já a perícia engloba os aspectos da lógica de programação, ou seja, as habilidades de decompor um problema e formar uma estratégia a longo prazo de resolução utilizando comandos isolados em uma determinada ordem para resolver o problema.

1.1 ENSINO DE PRINCÍPIOS DE OPERAÇÃO

No ensino de princípios de operação cada comando é visto isoladamente. Não há contextualização do mesmo. O objetivo a ser atingido é saber como escrever corretamente um comando e que ação ele executa, em outras palavras, seu significado.

As linguagens formais de programação para uso muito específico, como é o caso de um CDCT, geralmente são escritas por técnicos e engenheiros

especialistas na área de uma maneira *ad hoc*. Se de uma forma geral as linguagens de programação já não são criadas de uma maneira natural para a expressão textual, nesses casos há ainda o agravante de geralmente não existir uma interface de alto nível de operação. Sendo assim, apenas os operadores experientes conseguem atingir altos níveis de desempenho utilizando expressões textuais de baixo nível.

Adicionalmente, o cenário que se apresenta então é de linguagens não estruturadas e de pouca ou nenhuma reação (*feedback*), principalmente em se tratando de erros. No caso específico de uma CDCT, após verificar os "*logs*" de sessões de treinamento (alguns deles podem ser vistos no Anexo 3), foi possível perceber alguns problemas na linguagem de operação. Por exemplo, para erros distintos foi mostrada uma mesma mensagem de erro. Ou então mensagens parecidas acusam erros diferentes.

A central digital utilizada para testes (SPX2000) e todas as outras centrais apresentam uma grande rigidez sintática. Para tratar dessa grande rigidez e ajudar aprendizes, ela não é capaz de utilizar um número significativo de classes de erros. Estas poucas classes apresentam mensagens imprecisas e muito semelhantes entre si, dificultando sua utilização pelo iniciante.

Como a sintaxe de um comando é o primeiro grande obstáculo que um aprendiz encontra, um sistema tutorial que tenha por objetivo apoiar o ensino de programação deve se preocupar bastante com este tópico. É necessário prover um *feedback* sintático que explique melhor cada erro ou então implementar o ensino através de exemplos, onde o aprendiz pode escolher *templates* (máscaras) de comandos até se sentir confiante para escrevê-los sem ajuda.

Além do aspecto sintático, o ensino de princípios deve também ocupar-se em transmitir a semântica de cada comando. Somente sabendo com certeza o que cada comando faz é que o programador pode conseguir utilizá-los de maneira

correta para implementar a solução de um problema. Geralmente os sistemas oferecem exemplos de comandos executados, bem como um ambiente de livre exploração onde o aprendiz pode testar os comandos, verificar o resultado de cada comando até que tenha um conceito sólido e modelos mentais do que ocorre como efeito de cada comando.

1.2 ENSINO DA PERÍCIA DE OPERAÇÃO

Depois que os aprendizes sabem trabalhar com comandos isolados, é hora de aprender a colocá-los em conjunto para implementar soluções de problemas propostos. No caso da CDCT, um exemplo de problema pode ser cadastrar um assinante, que requer a seguinte seqüência de passos: (1) um comando para testar se um número telefônico escolhido está disponível, (2) um comando para testar se uma porta específica do hardware está livre para ser associada ao número telefônico escolhido, (3) um comando para associar o número escolhido à porta do hardware, (4) um comando para ativar o serviço da porta e (5) um comando para arquivar, em memória permanentemente, a inclusão do assinante.

Aqui não importa somente saber o que cada comando faz, mas também como colocá-los juntos e na ordem certa para fazer o que se espera que faça. A palavra chave portanto é ordem. Para que a solução do problema esteja correta, determinados comandos têm que ser executados na seqüência correta.

Estudos empíricos apontam que os programadores iniciantes apresentam grande dificuldade em integrar os comandos e conceitos para formar um programa. Algumas estratégias que um sistema para auxiliar o ensino de programação pode adotar para abordar este problema são (DIRENE; NASCIMENTO; PRETO; et al., 1996):

- visualização de cenários e concepção de micro-mundos;
- exemplos de cenários e soluções para que o aluno use tais exemplos como modelos para suas soluções.

Adquirindo perícia, o programador é capaz de formular soluções mentais para problemas utilizando modelos recolhidos por sua experiência. É possível abstrair a solução de um problema específico para um conjunto de sub-planos de soluções já vistas. É também importante salientar que essas soluções não dependem de sintaxe. Assim, uma vez adquirida a perícia, para que o aprendiz seja treinado em outra linguagem basta adquirir os princípios da mesma.

1.3 OBJETIVOS

O uso de um sistema tutorial como o apresentado neste trabalho para apoiar o ensino utilizando o computador permite que alguns objetivos sejam alcançados. Entre esses objetivos, pode-se destacar os seguintes:

- Diminuir o tempo necessário para o treinamento pois, além de um tutor humano, o aprendiz terá também o tutor eletrônico à sua disposição. Utilizando-se de ambos, o aprendiz pode ter sanadas suas dúvidas mais rapidamente. O tutor eletrônico está sempre disponível, atuando como um professor particular para o aprendiz.
- Aumentar o potencial de diagnóstico de erros. Através do tutor eletrônico, o aprendiz terá mais oportunidade de ter seus erros diagnosticados e corrigidos de uma maneira mais significativa do que a apresentada pela CDCT. Tendo a possibilidade de errar e recebendo apoio sintático e semântico mais amigável, o aprendiz tende a melhorar seu diagnóstico de erros.

- Facilitar o pré e pós-treinamento à distância. Um sistema tutorial pode ser levado a lugares remotos a um menor custo. O sistema tutorial também pode ser utilizado nas fases pré e pós-treinamento com um tutor humano para que o aprendiz possa ir se acostumando com o ambiente antes do treinamento, e possa praticar o que aprendeu na fase pós-treinamento.

1.4 VISÃO GERAL DO SISTEMA SATELIT

Este trabalho está inserido no sistema SATELIT (Sistema de Apoio à TELefonia com Inteligência integrada ao Treinamento) que, por sua vez, faz parte de um projeto experimental denominado PROTTEL-D (PROjeto de sistemas Tutoriais para TELefonia Digital). O SATELIT é um tutor baseado em simulação que permite interações ativas e passivas entre o operador e o ambiente (DIRENE, 1997).

Seu objetivo é apoiar o ensino de operação e manutenção de uma CDCT. Como a CDCT é operada por uma linguagem de programação, no SATELIT foram propostas soluções para que o sistema possa reagir tanto sintática como semanticamente.

O sistema tem embutido um simulador de uma CDCT (atualmente apenas o modelo SPX2000 está implementado). Este simulador apresenta reação (*feedback*) inteligente para erros sintáticos (princípios) cometidos pelo operador. A proposta e implementação de apoio sintático faz parte do escopo deste trabalho e é apresentada no decorrer com mais detalhes.

Para atacar o problema de perícia em programação, este trabalho apresenta uma proposta de solução (ainda não implementada). Uma vez que esta solução

esteja implementada, o aluno poderá contar com a reação do tutor não somente nos casos restritos a um comando, mas também ao longo de uma sessão completa.

A interface do SATELIT apresenta os seguintes elementos para seus usuários:

- terminal simulado: neste terminal (situado no alto à esquerda) o operador vê exatamente o que ocorreria numa CDCT real, ou seja, as mensagens de erro da central, as respostas afirmativas e o efeito de seus comandos. Seu objetivo é permitir que o operador se acostume com o ambiente em que vai trabalhar para que o treinamento não fique afastado da realidade.
- galeria de exercícios de exemplo: esta galeria (situada abaixo do terminal simulado) contém uma lista de exercícios típicos que um operador comumente vai executar. A galeria só está disponível quando o sistema está no modo passivo, e é acionada pelo aluno. Para que um item da galeria seja ativado basta que o aprendiz o selecione. Há botões de controle que permitem visualizar a execução passo a passo, recomeçar a execução ou mesmo simplesmente assistir a simulação do exercício.
- janela de mensagens: esta janela é mostrada ao aprendiz quando o sistema está no modo ativo. As mensagens de erro e de apoio prestadas pelo sistema são apresentadas nesta janela.

É interessante salientar que mesmo durante a execução passo a passo de um exercício, o aluno pode entrar comandos no terminal. O sistema passa então automaticamente para o modo ativo e analisa o comando, dando a reação pertinente (acerto ou erro). Além disso, o aluno também pode modificar o comando escolhido pelo apresentador passo a passo. Fazendo isso, o aluno estará executando outro exercício apenas baseado na galeria de exemplos.

1.5 ESTRUTURA DA DISSERTAÇÃO

Segue uma pequena exposição sobre a estrutura deste trabalho. O capítulo 2 procura mostrar o estado da arte em se tratando de ensino de programação apoiado pelo computador. Alguns sistemas são citados e as características que os fizeram importante são exploradas. O capítulo se estende e faz um apanhado de "*shells*" para ensino de conceitos, ressaltando que o ensino de programação não está entre os domínios onde essa facilidade foi aplicada.

Os capítulos 3 e 4 descrevem o trabalho realizado nesta dissertação. São mostradas as soluções propostas para o ensino de princípios e perícia, respectivamente. A parte que trata do ensino de princípios (capítulo 3) foi inteiramente idealizada e implementada. Ao longo do capítulo, a solução é mais detalhada, começando por sua arquitetura, onde cada ponto é explicado, até chegar ao tratamento de erros propriamente dito.

No capítulo 4 está a proposta ainda não implementada para dar apoio à lógica. Sua estrutura é análoga à do capítulo anterior. Primeiro a arquitetura do módulo é descrita. Depois de todos os elementos explicados, é exposto como será feita a utilização dos mesmos para o tratamento de erros de lógica.

A dissertação prossegue com uma pequena discussão sobre o tema. Algumas questões são levantadas e se procura avaliar as vantagens e desvantagens da solução implementada/proposta. E, finalmente, o trabalho é concluído com considerações sobre o que já foi feito e o que ainda é possível de se fazer. Logo a seguir estão os anexos. O primeiro contém a descrição da linguagem da SPX2000 em MELON (meta-linguagem utilizada pelo simulador). O segundo mostra as classes de erros sintáticos definidos com alguns exemplos. E o terceiro traz alguns extratos de sessões de treinamento na SPX2000 de onde foi possível extrair erros e analisar o comportamento tanto dos alunos quanto da central.

2 TRABALHOS CORRELACIONADOS

Este capítulo apresenta, predominantemente, uma revisão literária sobre o ensino de programação apoiado por computadores. Esta exposição começa com micromundos, prossegue com sistemas de diagnóstico de programas e ainda inclui *shells* inteligentes para o ensino de conceitos.

2.1 MICROMUNDOS PARA O ENSINO DE LINGUAGENS DE PROGRAMAÇÃO

Existem alguns sistemas já implementados para o apoio no ensino de linguagens textuais de programação. A maioria desses sistemas, como cita du Boulay (du BOULAY, 1988), tem se concentrado em apoiar os alunos na recuperação de erros lógicos ao invés de sintáticos ou semânticos. Mas como este trabalho trata separadamente dessas categorias, aqui eles serão expostos em itens distintos.

2.1.1 APOIO SINTÁTICO

O conhecimento sintático inclui esquemas sobre a forma como o código é escrito para linguagens textuais (du BOULAY; SOTHCOTT, 1987). Aparentemente, os aprendizes iniciantes tendem a acreditar que esta é a parte principal do aprendizado de programação, ao contrário dos educadores, que não vêem na sintaxe o principal tópico de enfoque (du BOULAY; SOTHCOTT, 1987). Isso provoca uma necessidade pedagógica direcionada para duas correntes de ação remediadoras: (1) contornar a rigidez sintática com ferramentas auxiliares de programação que mascarem tal rigidez e (2) atacar frontalmente a rigidez sintática com explicações precisas a respeito de erros cometidos.

De acordo com a primeira abordagem, geralmente o apoio sintático é feito pelos sistemas tutoriais através de editores de texto orientados a sintaxe ou então pela apresentação de máscaras (*templates*) onde o aluno só precisa preencher lacunas com seus dados. Os editores orientados a sintaxe são aqueles onde palavras-chave ficam em destaque ou então há a complementação de padrão (um *begin* gera um *end*, um *if* gera um *then* e assim por diante), por exemplo o **Alice Pascal** (TEMPLETON, 1986) usa um editor orientado a sintaxe. O sistema **Greaterp** (também conhecido por LISP Tutor) de Anderson (ANDERSON; BOYLE; REISER, 1985) é um exemplo de sistema que usa a técnica de *templates*.

Os sistemas que utilizam os editores orientados a sintaxe nem sempre resultam em aprendizagem para o aluno. Isso porque eles somente contornam a rigidez sintática sem efetivamente "ensiná-la" ao aluno. E muitas vezes tais sistemas não são construídos com fins educacionais. Esse tipo de apoio sintático é amplamente utilizado em compiladores comerciais, por exemplo.

O uso de máscaras também não resulta em aprendizado sempre. Esta técnica é de maior aproveitamento para programadores experientes (especialistas) do que para iniciantes. Um aluno iniciante muitas vezes não sabe com certeza o que uma máscara representa ou o que é esperado dele em tal situação. Já o programador experiente pode poupar seu tempo usando máscaras para preocupar-se mais com a parte semântica e lógica do trabalho.

O ataque frontal à rigidez sintática é importante e necessária no ensino a um aluno iniciante. Para que o aluno possa verificar a semântica de seu programa a sintaxe tem que estar correta. Portanto, quão mais cedo a sintaxe for dominada, melhor.

O *feedback* sintático ajuda o aluno iniciante a superar seu primeiro obstáculo no aprendizado de programação. Através deste *feedback* o aluno percebe

seus erros sintáticos e aprende como corrigi-los. Assim, é possível dizer que o valor educacional do *feedback* sintático é alto.

Porém, nenhum dos sistemas de ensino existentes para aprendizes iniciantes é capaz de oferecer explicações sobre erros sintáticos. Como este trabalho está voltado para alunos iniciantes, e devido ao alto valor educacional do *feedback* sintático, este tópico será tratado com a devida importância.

2.1.2 APOIO À LÓGICA POR VISUALIZAÇÃO

A visualização da execução dos programas é importante para o aprendiz iniciante. Esta importância advém das vantagens de mostrar ao aluno, de alguma forma, os efeitos internos produzidos durante a execução de seu código. Ramadhan e du Boulay (RAMADHAN; du BOULAY, 1993) argumentam que a visualização ajuda os aprendizes a entender como os comandos são executados e como o programa age dinamicamente. O aprendiz literalmente vê o que está acontecendo.

Entre os exemplos tradicionais de visualização de programas pode-se citar a animação de algoritmos e as alterações de comportamento das células de memória correspondentes aos objetos ou variáveis do programa do aprendiz. A animação de algoritmos demonstra, por meio de metáforas gráficas, o que está acontecendo com abstrações das variáveis, as quais sofrem diversas variações de valor ao longo do tempo. Através da animação, o aprendiz pode assimilar melhor o significado de seu código com a ajuda de vários meios de comunicação, os quais são predominantemente visuais e sonoros.

Quando é feita a animação de algoritmos utilizando-se poucos meios de comunicação, têm-se mecanismos equivalentes a depuradores. O aprendiz pode verificar como as variáveis são alocadas, além de ficar claro o caráter dinâmico da memória. Em escala reduzida, por exemplo, é possível saber realmente o que acontece quando uma atribuição de valor é feita.

Szabo e Poohkay (SZABO; POOHKAY, 1995) comentam que as imagens e gráficos aumentam a quantidade de material aprendido tanto por adultos como por crianças. Além disso, estes autores também destacam o fato de que pesquisas feitas mostram que o uso de imagens resulta em altos níveis de satisfação do aluno perante o aprendizado. Como fator de apoio semântico, a importância desse efeito positivo se dá pela representação de grandes quantidades de significados de conceitos utilizando formas gráficas, amenas ao aparelho visual ou auditivo.

2.2 SISTEMAS DE DIAGNÓSTICO DE PROGRAMAS

Quando se fala em ensino de programação, deve-se ter em mente a necessidade de analisar programas para determinar se estão corretos ou não do ponto de vista da lógica de programação. Para um tutor humano, essa é uma tarefa razoavelmente simples, que é desenvolvida quase como conseqüência do aumento da experiência profissional. Mas quando se trata de um sistema tutor automático essa tarefa é mais complicada pois é preciso codificar explicitamente todo o conhecimento pedagógico de como explicar elementos de conhecimento do próprio domínio de programação em uma linguagem.

Uma decisão que deve ser tomada desde as primeiras fases do projeto de um tutor automático é se o sistema de diagnóstico fará a análise durante o evento de construção do programa, ou seja, enquanto o aprendiz está desenvolvendo o programa, ou após o evento, quando o programa já está completo. Nos próximos sub-itens são mostradas as características de cada opção e os problemas com que elas têm de lidar.

2.2.1 DIAGNÓSTICO COM ANÁLISE PÓS-EVENTO

Este tipo de diagnóstico depende de ter o programa, a tarefa pronta. Se por um lado não é necessário preocupar-se com a incompleteza de dados, é necessário levar em conta a existência de erros múltiplos na programação. A existência de vários erros é comum em programas de iniciantes. E aumenta consideravelmente a complexidade do sistema que está fazendo o diagnóstico. Em compensação, este modo de diagnóstico dá mais liberdade ao aprendiz. Uma vez que apenas ao final do programa ele será submetido a uma análise pelo tutor, o aprendiz pode tentar inventar soluções mais criativas. Ou então explorar caminhos infrutíferos mas relevantes do ponto de vista pedagógico.

Alguns exemplos clássicos de tutores para programação com diagnóstico pós-evento são LAURA (ADAM; LAWRENT, 1980), PROUST (JOHNSON, 1986) e MYCROFT (GOLDSTEIN, 1975). O sistema LAURA, como aponta Wenger (WENGER, 1987), pega um programa em FORTRAN feito por um estudante e um programa resposta e tenta fazer uma correspondência entre eles através de uma série de transformações. O primeiro passo é transformar ambos os programas em grafos. Depois o modo como as variáveis são declaradas e usadas é padronizado. Finalmente, os grafos são padronizados. Aí sim é tentado o casamento entre os grafos. Quando há erros no programa do estudante, o sistema tenta explicar as discrepâncias em termos de erros simples como nome errado de variável, por exemplo. Se isto não é possível, o sistema mostra o que não está conseguindo casar na esperança de que o usuário possa corrigir o problema.

PROUST é um sistema que pode detectar uma grande variedade de erros lógicos em programas escritos em Pascal por estudantes. O sistema tem disponível um repertório de planos para atingir metas. Ele sistematicamente tenta casar os vários planos com o código do estudante apresentado a ele. O sistema tenta entender as discrepâncias deformando seus planos de acordo com um conjunto de

bugs (erros) previamente catalogados. O sistema escolhe o casamento que minimiza o número de violações hipotéticas e mostra ao usuário. Contudo, o sistema pode ser facilmente enganado com códigos equivalentes mas não idênticos aos esperados. Outro problema é que o sistema não atua interativamente com o usuário e também não pode mostrar exatamente ao estudante o que acontece na execução do código sob diferentes circunstâncias.

O sistema MYCROFT pode detectar e reparar erros em programas em LOGO (PAPERT, 1980) feitos para desenhar figuras com linhas. A especificação do que o programa pretende fazer é chamado de modelo. O MYCROFT começa o processo de detecção de erros executando o programa do estudante simbolicamente e anota cada linha de código com comentários que guardam as mudanças de estado da tartaruga (o objeto usado para desenhar) e as propriedades de qualquer forma que foi desenhada.

2.2.2 DIAGNÓSTICO COM ANÁLISE DURANTE O EVENTO

Este tipo de diagnóstico é efetuado enquanto o programador aprendiz está escrevendo seu programa. Em que ponto o sistema fará intervenção depende da granulação imposta pelos idealizadores do mesmo. Alguns sistemas fazem o acompanhamento caracter a caracter, por exemplo. E outros esperam para intervir até que uma palavra ou comando esteja completo.

Essa intervenção é baseada na tentativa de casamento da resposta apresentada pelo aprendiz com um modelo pré-formado da solução ou então com regras. A esta técnica de fazer o aluno seguir um modelo de resposta foi batizada por Anderson (ANDERSON, 1984) de *model tracing*.

A técnica do *model tracing* não deixa que o aluno se afaste muito de um caminho correto. Por causa disso, quase como um subproduto do *model tracing* durante o evento obtém-se a resposta imediata do tutor (*immediate feedback*). Isso

leva ao fato do diagnóstico preocupar-se com um único erro de cada vez, o que minimiza o problema do não determinismo e, conseqüentemente, de implementação.

Mas há outras razões para a intervenção imediata. Como aponta Anderson em seus estudos (ANDERSON; BOYLE; CORBETT; et al., 1990), há evidências psicológicas de que *o retorno no momento* do erro é efetivo pois é mais fácil para o estudante analisar o que levou ao erro e fazer a correção apropriada. Além disso, evita que o estudante fique vagueando por caminhos (soluções) incorretos. Du Boulay (du BOULAY; SOTHCOTT, 1987) aponta ainda outra razão defendida por Anderson de que se for permitido ao aluno persistir nos erros, ele fatalmente aprenderá o método errado ao invés do certo.

2.3 SHELLS PARA ENSINO DE CONCEITOS

Apesar de os Sistemas Tutoriais Inteligentes (STIs) já existirem há algum tempo, eles nunca foram amplamente utilizados nos estabelecimentos de ensino. Algumas razões para isso foram apontadas pelos principais pesquisadores de IA na Educação. Entre elas podem ser citadas as seguintes: 1) eles não permitem ao professor alterar o modo como os conceitos são ensinados e 2) eles se restringem a um único domínio. Para resolver estes problemas foram criados os sistemas de autoria e as *shells* de ensino. Estes assuntos serão detalhados separadamente nos tópicos a seguir.

2.3.1 SISTEMAS DE AUTORIA

Através de entrevistas com professores, Major e Reichgelt (MAJOR; REICHGELT, 1991) identificaram que uma das razões pelas quais os STI´s não são

amplamente utilizados em salas de aulas é porque a maioria dos STIs ensina de acordo com uma estratégia fixa de ensino. Isso significa que eles não permitem que os professores alterem o modo como o conteúdo é ensinado.

Os sistemas de autoria surgiram para permitir que os professores determinem como os conceitos serão ensinados aos alunos através de um STI. Através de um sistema de autoria os professores podem especificar "cursos" diferentes para alunos diferentes usando os mesmos conceitos e o mesmo sistema tutorial.

Murray e Woolf (MURRAY; WOOLF, 1990) apontam que para que os STI´s deixem de ser produtos utilizados apenas pela comunidade de pesquisa, os professores têm que estar envolvidos no desenvolvimento dos mesmos. O fato do envolvimento dos professores ser crucial durante o desenvolvimento de um STI também é levantado por Blessing (BLESSING, 1997). Ele diz que os professores não conseguem desenvolver um STI sozinhos pois é preciso ter conhecimentos fora de sua área de domínio.

A existência de um sistema de autoria apresenta mais de uma vantagem. Além de dar uma maior flexibilidade ao professor na hora de definir seu curso, também aumenta a velocidade de desenvolvimento de um STI, como apontam Major, Ainsworth e Wood (MAJOR; AINSWORTH; WOOD, 1997).

Hoje existem sistemas de autoria para diferentes domínios. Entre eles podemos citar COCA (álgebra) (MAJOR; REICHGELT, 1991), Demonstr8 (aritmética) (BLESSING, 1997), RUI (radiologia) (DIRENE, 1990). Já existem inclusive sistemas de autoria utilizando hipermídia, como o SASHE (NUNES; HASEGAWA; VIEIRA; et al., 1997), por exemplo.

2.3.2 *SHELLS*

Inicialmente cada sistema especialista ou STI era criado a partir do nada. Com o tempo, verificou-se que todos tinham muito em comum (RICH; KNIGHT, 1994). Em 1976 Shortliffe demonstrou ser possível separar o conhecimento específico do domínio de seu interpretador de diagnóstico (SHORTLIFFE, 1976). A partir de então começaram a surgir as *shells*, ou ferramentas independentes de domínio. Os usuários das *shells* devem implementar apenas a base de conhecimento do domínio específico, ao invés do sistema inteiro. Com isso, o código da *shell* pode ser reutilizado tantas vezes quanto necessário.

A implementação de *shells* aumenta a velocidade com que um STI é desenvolvido além de abaixar seu custo. Esse fato pode vir a aumentar a utilização de STI´s uma vez que o único esforço no intuito de desenvolver um novo STI é a implementação de seu conteúdo específico propriamente dito.

Derivado do MYCIN (SHORTLIFFE, 1976) surgiu a *shell* EMYCIN (Empty MYCIN) (BUCHANAN; SHORTLIFFE, 1984) acoplada ao tutor GUIDON (CLANCEY, 1983; 1987) para o ensino do diagnóstico para domínios variados de doenças infecto-contagiosas. E muitas outras *shells* na área de STI foram desenvolvidas desde então como, por exemplo, o BIP (BARR; BEARD; ATKINSON, 1976), BYTE (BONAR, 1985), Micro-SEARCH (SLEEMAN, 1987a), LMS/PIXIE (SLEEMAN, 1982; 1987b). Essas *shells* são aplicadas a vários domínios, mas nenhuma delas é usada para o ensino de programação de computadores. Os sistemas tutores e micromundos para o ensino de programação são específicos para um único domínio. Assim, sempre que for preciso ensinar uma nova linguagem, por exemplo, é necessário implementar um sistema tutor desde o início.

2.3.3 SHELLS PARA SISTEMAS NÃO INTELIGENTES

Independente do fato de um sistema de ensino ser inteligente ou não, ele pode utilizar uma *shell*. O que determina o uso de uma *shell* é a filosofia de desenvolvimento de sistemas adotada.

Como apontam Nicolson e Scott (NICHOLSON; SCOTT, 1986), há duas estratégias para produzir um *software*. Uma delas é a produção individual e a outra em grupo. Na produção individual a vantagem é que um único programador/professor pode desenvolver um sistema tutor rapidamente. Geralmente ele utiliza um ambiente de autoria e desenvolve cada sistema de forma bem rápida. Em compensação, usualmente a qualidade destes sistemas é questionável.

Ao contrário, quando os sistemas são desenvolvidos por uma equipe, geralmente a qualidade é maior, mas a quantidade é menor. Numa equipe cada um é responsável por um estágio do desenvolvimento. Primeiro os professores determinam qual o conteúdo do sistema bem como a forma pela qual este conteúdo será ensinado ao aluno. Em seguida um programador codifica essas especificações. O programa então é testado com alunos.

Essa divisão de tarefas, deixando cada um trabalhar de acordo com sua especialidade (o professor "ensina" e o programador codifica especificações), faz com que os sistemas desenvolvidos tenham uma melhor qualidade. Em contrapartida, o número de sistemas produzidos diminui devido às interações entre cada fase.

3 IMPLEMENTAÇÃO DO APOIO SINTÁTICO NO SISTEMA SATELIT

Neste capítulo será descrito o que foi executado no sistema SATELIT como parte deste trabalho para o apoio à aquisição de conhecimento de princípios de operação, mais especificamente, o apoio na recuperação de erros sintáticos. Primeiro será apresentada a arquitetura do Modelo de Apoio a Erros de aspecto Sintático (MAES). Em seguida a descrição de como é feito o tratamento de erros, encerrando com alguns exemplos de aplicação.

3.1 ARQUITETURA DO MAES

O MAES é composto dos elementos mostrados na Figura 1. Cada um destes componentes será agora descrito em mais detalhes.

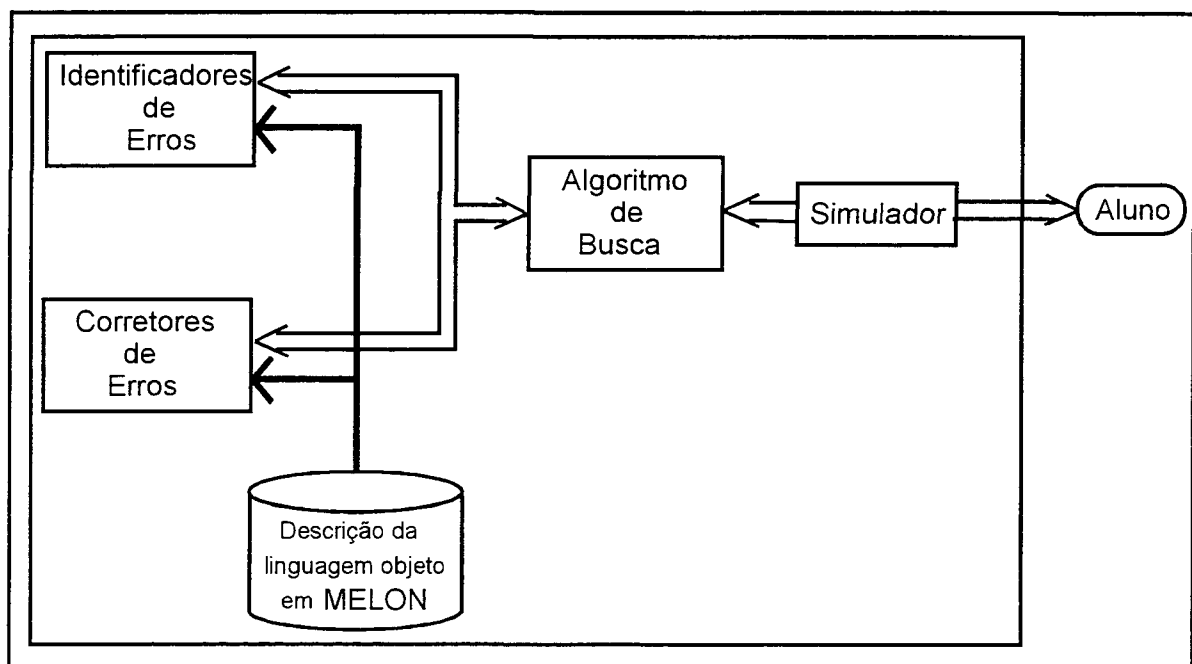



Figura 1 - **Estrutura do MAES**

3.1.1 DESCRIÇÃO DA LINGUAGEM OBJETO DE OPERAÇÃO EM META-LINGUAGEM MELON

Sendo o objetivo do projeto do qual este trabalho faz parte construir uma ferramenta genérica (*shell*) para o ensino de programação da operação de um conjunto variado de CDCTs, é necessário o uso de uma meta-linguagem para representar a linguagem objeto de operação de uma central de comutação específica. Esta meta-linguagem foi denominada MELON (MEta-Language for defining OperatioN commands) (DIRENE, 1997) e foi utilizada para descrever a linguagem objeto da CDCT SPX2000. A Figura 2 mostra alguns comandos da central descritos em MELON.

MELON é uma variação de BNF, uma meta-representação amplamente conhecida. Portanto, ela faz uso de meta-símbolos, símbolos não-terminais e símbolos terminais. Os meta-símbolos com seus significados respectivos adotados para a MELON são os seguintes:

=> produz

| ou-exclusivo

{ } repetição indefinida

[ ] opcionalidade internamente a um comando ou parâmetro

Para símbolos não-terminais convencionou-se que devem começar e terminar por um sublinhado (*underscore*) como, por exemplo, "_valStn_" e "_valPen_". Além desses, que são definidos de acordo com mnemônicos da linguagem objeto, há alguns símbolos não-terminais especiais, definidos na própria MELON, para representar dígitos decimais (0 a 9) e hexadecimais (0 a F) que são, respectivamente, os símbolos "$" e "@". Há ainda um símbolo para indicar um espaço em branco definido por um traço "-".

Qualquer outro símbolo é tratado pela MELON como símbolo terminal da linguagem objeto. Por convenção, são escritos em letras maiúsculas. No caso da

linguagem específica da CDCT SPX2000, alguns desses símbolos são: "ADD", "SAV", "STNASSN", ":" , "#", "0". Outros símbolos terminais podem ser vistos no Anexo 1, onde se encontra a listagem completa da linguagem da CDCT SPX2000, descrita em MELON. Essa meta-linguagem será utilizada para a definição de heurísticas independentes da linguagem de operação que se prestam à detecção e correção de erros sintáticos nos comandos dados pelos aprendizes.

```
ADD STNASSN: STN#"=' _valStn_, TYPE '=' _valType_, CAT# '='
_valCat_, PEN# '=' _valPen_, COS# '=' _valCos_ [ , ORIG '=' _valOrig_ ] [
, EFLG '=' _valEflg_ ] [ , RSTR '=' _varRstr_ ] [ , RSCT '=' _val_Rsct_ ] [,
WARM '=' _valWarm_ ] [ , IDX# '=' _valIdx_ ] ;
        _valStn_ => $$$$
        _valType_ => DP | DTMF | HOTLINE
        _valCat_ => 1 | 2 | 3 | 4 | 5 | 6 | 7
        _valPen_ => $$$@
        _valCos_ => $ | $$
        . . .
 CHA STNASSN: OSTN '=' _valOstn_ [ , STN# '=' _valStn_ ] . . . ; ]
        _valOstn_ => $$$$
        _valChbu_ => Y | N
        . . .
 SAV CUSTDATA [ : SYST '=' _valSyst_ ] ; ]
        [ _valSyst_ [ ACTIVE ] [ STANDBY ] [ BOTH ] ]
```

Figura 2 - **Comandos escritos em MELON**

Através da análise de arquivos do histórico de sessões reais de treinamento para o uso da CDCT SPX2000, foi possível classificar determinados tipos de erros. Estes erros identificados foram então divididos em classes e pôde-se implementar rotinas identificadoras e corretoras para eles.

3.1.2 IDENTIFICADORES DE ERROS

As rotinas identificadoras de erros foram projetadas como parte deste trabalho para a detecção de erros sintáticos em um comando. Cada rotina é

responsável por "analisar" um comando e procurar por um único e específico tipo de erro.

Para lidar com este problema de detecção de erros, foram criadas aproximadamente 40 classes de erros sintáticos e, para cada uma delas, um identificador de erro foi implementado. Estas classes têm por objetivo dar mensagens mais específicas ao operador. Alguns exemplos destas classes são:

- **erro ortográfico sem aproximação**, que detecta um erro ortográfico em nome de comando, complemento, ou mesmo em parâmetro ao qual não foi possível associar um símbolo por semelhança léxica (para símbolos que são seqüências de caracteres alfa-numéricos);
- **erro ortográfico com aproximação**, que detecta um erro ortográfico em nome de comando, complemento, ou mesmo em parâmetro onde é possível associar um símbolo por semelhança léxica;
- **incompatibilidade de objeto com relação ao comando**, que detecta o uso de termo (alfa-numérico) incompatível com o comando corrente, onde um outro termo deverá ser aconselhado para substituição;
- **uso de parâmetro de outro comando**, que detecta a aplicação de parâmetro de outro comando mas que não se aplica ao comando corrente;
- **substituição de símbolos devido à semelhança visual**, que detecta a substituição indevida de símbolo (não alfa-numérico) por outro símbolo (não alfa-numérico) existente na linguagem, devido à semelhança visual entre eles. Por exemplo: ":" ";" "|" ou "/" "\" ou "0" "O" etc;
- **inversão (2 a 2) de valores exclusivos de parâmetros**, que detecta um erro por inversão de valores de parâmetros, onde ambos os valores envolvidos são exclusivos (2 a 2) na comparação do universo de valores possíveis da definição dos parâmetros;

- **caracteres inválidos em valores de parâmetros**, que identifica erros específicos de caracteres inválidos em valores de parâmetros, tais como: letra fora da faixa de símbolo hexadecimal, troca de "0" por "O" (e vice-versa), e outros;
- **repetição de parâmetro no comando**, que detecta a digitação repetida de um dado parâmetro, mantendo ou não seus valores idênticos;
- **uso de parâmetros onde não há**, que detecta a tentativa de fornecer parâmetros para um comando que, sem tais parâmetros, já formaria um comando completo;
- **uso de nome de comando como valor de parâmetro**, que detecta a substituição indevida de um valor de parâmetro por um nome de comando existente na linguagem, devido à semelhança de nomes ou outra razão qualquer.
- **uso de nome de parâmetro como comando**, que detecta a substituição indevida de um comando por um nome de parâmetro existente na linguagem, devido à semelhança de nomes ou outra razão qualquer.

Se um comando apresentar mais de um erro (sejam eles do mesmo tipo ou não), cada rotina identificadora de erros será capaz de apontar apenas um. Por exemplo:

- DIS STNASSN STN# = 7700

Este comando apresenta dois erros (a falta do símbolo *":"* entre "STNASSN" e "STN" e a falta do delimitador *";"* ao final do comando). Sendo assim, se o comando-exemplo acima for submetido ao identificador de erros por falta de separador, tem-se um retorno afirmativo, situação semelhante que ocorrerá com o identificador de falta do delimitador de final de comando. Porém, se este comando for aplicado ao identificador de erros ortográficos (ou tipográficos), este identificador

retornaria que não há nenhum erro. Somente quando este comando for submetido aos identificadores de falta de separadores e falta de delimitador é que um erro será acusado.

Às vezes o comando apresenta um erro que o operador não consegue detectar ou então corrigir sozinho. No comando a seguir há a falta do símbolo terminal obrigatório '#' depois do parâmetro PEN. Mas como em outros comandos este parâmetro (PEN) não requer o uso do símbolo '#', dificilmente o operador iniciante conseguirá detectar e/ou corrigir este erro sem a ajuda de um tutor. Por exemplo:

- ADD STNASSN : STN# '=' 8888, TYPE '=' DP, CAT# '=' 2, PEN '=' 717F;

Um outro exemplo desta dificuldade na detecção/correção pelo operador pode ser vista em:

- CHA SERVICE: MODE = I, TYPE = X, EXTEN = **700F**;

Explicação do erro: **700F** é um provável número de porta por causa da composição do *hardware*, quando o que deveria ser aplicado é o número do assinante.

No Anexo 2 encontra-se a descrição completa de todas as classes de erros com exemplos de ocorrência.

3.1.3 CORRETORES AUTOMÁTICOS DE ERROS

Para algumas das classes identificadoras de erros foram criados corretores automáticos correspondentes. Os corretores implementados servirão para a transformação de um estado em outro no algoritmo de busca que será usado para a detecção de erros. Eles não foram idealizados e implementados com o intuito de apresentar simplesmente o comando corrigido ao aluno, uma vez que a prática da correção automática não teria efeito pedagógico positivo perante o aprendiz. No

entanto, as correções efetuadas são guardadas internamente e opcionalmente podem ser mostradas ao aluno.

Nem todos os erros são facilmente corrigidos automaticamente. Por isso foram escolhidas apenas algumas classes identificadoras de erros para a implementação de corretores correspondentes tais como:

- **erro ortográfico**, que corrige um erro ortográfico em nome de comando, complemento, ou mesmo em parâmetro de um comando (símbolos que são seqüências de caracteres alfa-numéricos);
- **esquecimento de símbolos comumente usados como separadores**, que corrige um erro por esquecimento de símbolos (não alfa-numéricos) que, em diversas linguagens, são freqüentemente utilizados como separadores. Exemplo, "," ";":";
- **inversão (2 a 2) de valores de parâmetros**, que corrige um erro por inversão de valores que são exclusivos de um dado parâmetro, colocados em outro parâmetro, tendo sido um valor "típico" (possível) do primeiro colocado como valor deste segundo (e vice-versa);
- **contração de símbolos**, que corrige a não inclusão de espaços em branco entre dois ou mais símbolos (ou termos) válidos da linguagem;
- **inversão da ordem de símbolos/termos consecutivos**, que corrige a inversão da ordem em que dois símbolos/termos aparecem em um comando;

Cada corretor também é construído de forma independente da linguagem objeto de operação e seu funcionamento é regido pela descrição da linguagem objeto. O que essas classes fazem é simplemente um casamento de padrões entre um comando bem formado e o recebido do operador. Quando possível, é feito o preenchimento das lacunas existentes.

3.1.4 ALGORITMO DE BUSCA

O elo de ligação entre o simulador da SPX2000 e a identificação e correção de classes de erros é efetuado por um algoritmo de busca. Esse algoritmo é também responsável pela imposição de ordem de prioridade entre as classes. Na seção 3.2, onde será mostrado como foram tratados os erros nos comandos, ficará mais claro o porquê dessa ordem imposta à aplicação de identificadores e corretores de classes de erro, assim como o contexto de aplicação da busca heurística.

Na hora de decidir por um algoritmo de busca heurística para melhorar o desempenho do MAES, vários algoritmos foram considerados. A escolha foi feita devido a uma característica do atual problema: o estado final da busca não é conhecido. A chave para a busca é um comando (uma cadeia de caracteres), que pode ou não estar sintaticamente errado. Se está errado, não é possível, de imediato, determinar a qual comando ele corresponde.

Devido a essa característica, foi escolhido um algoritmo de busca com encadeamento para frente. E depois de alguns testes iniciais bem sucedidos ficou decidido que, dentre os algoritmos de busca que se encaixam nesta classe, o escolhido seria o de busca pela melhor escolha (*best first*). Esse algoritmo parece se adequar à solução do problema. Além disso, sua implementação é simples.

O algoritmo de busca pela melhor escolha faz uso de uma função heurística para determinar seu próximo passo. É através dessa função que o algoritmo sabe se encontrou um estado meta ou então qual o próximo nó da árvore de busca é mais promissor para análise.

Cada nó da árvore de busca é composto dos seguintes elementos:

1. comando: uma cadeia de caracteres contendo o estado atual do comando originalmente digitado pelo aprendiz;

2. número de erros: um número inteiro que indica a quantidade de erros encontrados no comando deste nó;
3. mensagem: cadeia de caracteres contendo a mensagem de erro composta por todas as mensagens acumuladas a partir da identificação e correção automática de erros. Assume-se que quanto maior a mensagem, maior o número de correções efetuadas no comando.

A Figura 3 mostra graficamente a estrutura de um nó (a) da árvore de busca bem como um nó exemplo (b).

(a) Estrutura de um nó **(b)** Exemplo de um nó

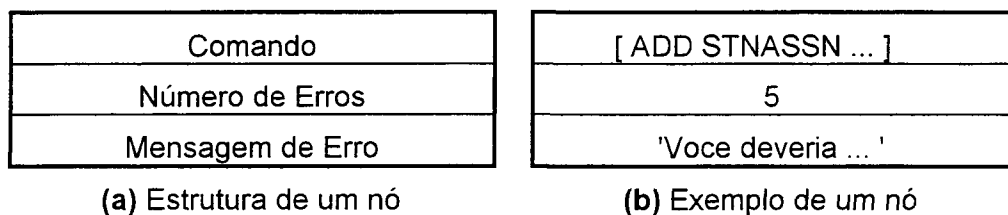



Figura 3 - **Estrutura e exemplo do nó da árvore de busca de apoio sintático**

O objetivo do algoritmo é o de minimizar o número de erros no comando, explorando primeiro os ramos mais curtos da árvore de busca para também minimizar o número de mensagens de erros a serem apresentadas aos alunos. Uma vez que se quer encontrar um comando sintaticamente correto, quanto menor o número de erros ele apresentar, melhor. E em caso de dois ou mais nós empatarem no número de erros, o algoritmo opta por aquele que sofreu maior número de correções.

A função heurística usa como parâmetros o número de erros que o comando apresenta e o número de correções que já foram feitas no comando. O campo com o número de erros é o responsável pela parada do algoritmo de busca. Quando seu valor é 0 significa que o comando está correto ou que foi totalmente corrigido. Se for infinito (representado por um número absurdamente alto, o maior número possível), o comando ainda apresenta erros que não puderam ser corrigidos. Ambas as situações são condições de parada.

o comando ainda apresenta erros que não puderam ser corrigidos. Ambas as situações são condições de parada.

O campo mensagem do nó é preenchido pelas classes identificadoras de erros. A mensagem é retornada para o operador como apoio, além do comando corrigido. Quando o campo mensagem está vazio significa que o comando do operador está sintaticamente correto.

3.1.5 O SIMULADOR

Em paralelo a este trabalho, foi implementado um simulador da CDCT SPX2000 por outras pessoas pertencentes ao grupo de pesquisa. Este simulador faz acesso ao MAES, deixando essa tarefa transparente para o operador. É no simulador que o operador tem a chance de praticar seus conhecimentos e receber a ajuda sobre aspectos sintáticos necessária para melhorar seu desempenho inicial. O simulador recebe o comando dado pelo operador e chama o algoritmo de busca. Se este retornar alguma mensagem de erro, o SATELIT a mostra ao operador, bem como o comando corrigido, quando possível. Uma descrição mais detalhada desse simulador pode ser encontrada em Outi (OUTI, 1997).

A seguir será mostrado o que foi feito para o tratamento de erros em comandos.

3.2 TRATAMENTO DE ERROS

Nesta seção será comentado quais os procedimentos tomados por este trabalho para o tratamento de erros sintáticos em comandos. Um único comando pode apresentar apenas um ou vários erros. Usando esta divisão para classificar os

3.2.1 ERROS SINGULARES

As rotinas identificadoras e corretoras de erros foram feitas para lidar com apenas um erro. Então no caso de comandos com um único erro é suficiente submeter o comando a cada rotina até que alguma delas seja capaz de identificar e, se possível, corrigir o erro.

A correção desse tipo de erro é relativamente fácil. Basta aplicar a rotina corretora correspondente, quando houver. Um único passo de repetição onde o comando é aplicado a uma rotina de cada vez é suficiente para lidar com os comandos com um único erro.

A Figura 4 mostra a resposta da rotina implementada a um comando com um único erro. A primeira rotina (a) apenas diagnostica o erro. E a segunda (b) além de detectar o erro, faz a correção quando há uma classe corretora disponível.

(a) *('DIS SERVCE;') =>*
*** Voce provavelmente cometeu um erro de grafia. Voce digitou SERVCE onde acreditamos ser o correto a palavra SERVICE.*

(b) *('DIS SERVCE;') =>*
*** DIS SERVICE ;*

Figura 4 - **Resposta da rotina de detecção e correção de erros sintáticos**

3.2.2 ERROS MÚLTIPLOS

Quando um único comando apresenta mais de um erro, diz-se que este comando apresenta erros múltiplos. Porém, há casos em que um único erro pode ser enquadrado em mais de uma classe, o que leva aos falsos erros múltiplos.

3.2.2.1 ERROS MÚLTIPLOS VERDADEIROS

Para lidar com os erros múltiplos verdadeiros, ou seja, onde há vários erros e cada um deles pode ser tratado como um erro singular, não foi preciso mudar em

nada a rotina implementada. A busca em profundidade simples foi suficiente. Em outras palavras, a reaplicação sistemática de passos simples de identificação e correção automática até que o sistema chegue internamente a uma expressão sintaticamente correta. Quando isto ocorre, basta divulgar para os treinandos as mensagens de erro correspondentes que foram acumuladas durante a busca em profundidade.

A Figura 5 mostra o exemplo de um comando com erros múltiplos verdadeiros. A primeira rotina (a) apenas faz o diagnóstico dos erros e em seguida a outra rotina (b) faz, além do diagnóstico, a correção dos mesmos.

(a) ('DIS SERVICE:EXTN=1000 1015') =>
*** Faltou ; no final do comando. Voce colocou um espaco em branco separando 10001015 em 1000 e 1015. Para que o comando fique correto, retire este espaco em branco da sua expressao.*

(b) ('DIS SERVICE:EXTN=1000 1015') =>
*** DIS SERVICE : EXTN = 10001015 ;*

Figura 5 - **Resposta da rotina para detecção e correção de múltiplos erros sintáticos**

3.2.2.2 FALSOS ERROS MÚLTIPLOS

Um comando com falsos erros múltiplos é aquele em que há apenas um erro, mas que acaba sendo enquadrado em várias classes. Esses erros são chamados "dependentes". Como só há um único erro, os outros são apenas uma reinterpretação ou conseqüência dele.

Aqui está um exemplo de comando que pode ser classificado neste tipo:

*ADD **STNASN**: STN# = 8861, TYPE = DP, CAT# = 1, PEN# = 700D, COS# = 0;*

Este comando apresenta os seguintes erros quando submetido às rotinas identificadoras:

- STN concatenado com ASN : STN é um parâmetro válido para o comando ADD, e aqui há a possibilidade dele estar concatenado com a palavra ASN. A rotina interpreta que encontrou o parâmetro STN e que este está inadvertidamente concatenado com o próximo símbolo ASN (não identificado).
- STN# não é parâmetro válido : como STNASN não é um objeto válido para o comando ADD, não há uma regra de boa formação e portanto o parâmetro STN# não é reconhecido como válido para esta combinação comando-objeto.
- STNASSN como aproximação de STNASN : existe um objeto para o comando ADD chamado STNASSN que pode ser uma aproximação ortográfica para STNASN.

O último é, na verdade, o responsável real pelo erro e é simplesmente um problema ortográfico.

Para tratar esse tipo de erro é necessário estabelecer uma ordem de prioridade na aplicação dos identificadores de classes de erros. Os erros mais comuns, como por exemplo a falta de delimitador no final do comando, ou mesmo um erro ortográfico devem ser tratados (identificados e internamente corrigidos) o quanto antes pelo algoritmo de busca para melhorar seu desempenho.

A busca em profundidade simples continuou a resolver o problema, mas em alguns casos o resultado deixou a desejar. Assim, como melhoria, foi implementado um algoritmo de busca heurística pela "melhor escolha" (RICH; KNIGHT, 1994), descrito na seção 3.1.4. No próximo tópico serão mostrados alguns exemplos de aplicação do algoritmo.

3.3 EXEMPLOS DE APLICAÇÕES REAIS

Esta seção vai mostrar dois exemplos de aplicações do MAES integrado com o simulador da CDCT SPX2000 no sistema SATELIT. Nos exemplos, os comandos dados pelo aprendiz apresentaram erros que foram apontados de duas maneiras. Na tela do simulador a mensagem de erro é exatamente a mensagem que a CDCT mostra a seus operadores. A mensagem produzida pelo MAES é mostrada em uma janela de diálogo à parte, para que tenha destaque para o operador.

O primeiro exemplo de erro ocorre durante a execução do seguinte enunciado:

> É necessária a alocação de 3 (tres) novos assinantes em uma central que possui apenas 8 assinantes já implantados. Para atingir este objetivo, os seguintes passos devem ser efetuados:
>
> 1 - Determinar um numero de chamada para o assinante,
> 2 - Determinar uma porta no hardware para alocar o assinante,
> 3 - Associar o numero do assinante à porta determinada,
> 4 - Colocar a porta do hardware em serviço.

A situação onde ocorre o erro pode ser visualizada na Figura 6. Na parte inferior da figura podem-se ver dois quadros onde estão os enunciados dos exercícios propostos. O aprendiz pode escolher um deles para tentar responder. Acima há três botões (Apresenta, Passo a Passo e Recomeçar), que o aprendiz pode utilizar para ver o simulador executar a solução do enunciado escolhido. Acima dos botões fica o "monitor" da central tal qual é visto pelo operador. A pequena janela de fundo cinza vista à direita é a janela de erros encontrados no comando corrente.

No momento em que a figura foi tomada, um erro havia ocorrido. No terceiro comando dado pelo aprendiz, quando ele digita o seguinte comando:

ADD STNASSN: STN=8889, TYPE = DP, CAT# = 1, PEN# 700D, COS# = 0;

é que foi detectado o erro.

Figura 6 - **Snapshot do sistema SATELIT**

Neste comando, como a mensagem de erro mostra, está faltando o símbolo # que é obrigatório depois do objeto STNASSN. Mas a única mensagem de erro da central para o operador é que foi encontrado um identificador inválido pois um símbolo # era esperado e foi encontrado o símbolo :. Para o operador iniciante a mensagem de identificador inválido não significa muita coisa. E, provavelmente, algum tempo extra será perdido até que se perceba a falta do símbolo # após o objeto STNASSN.

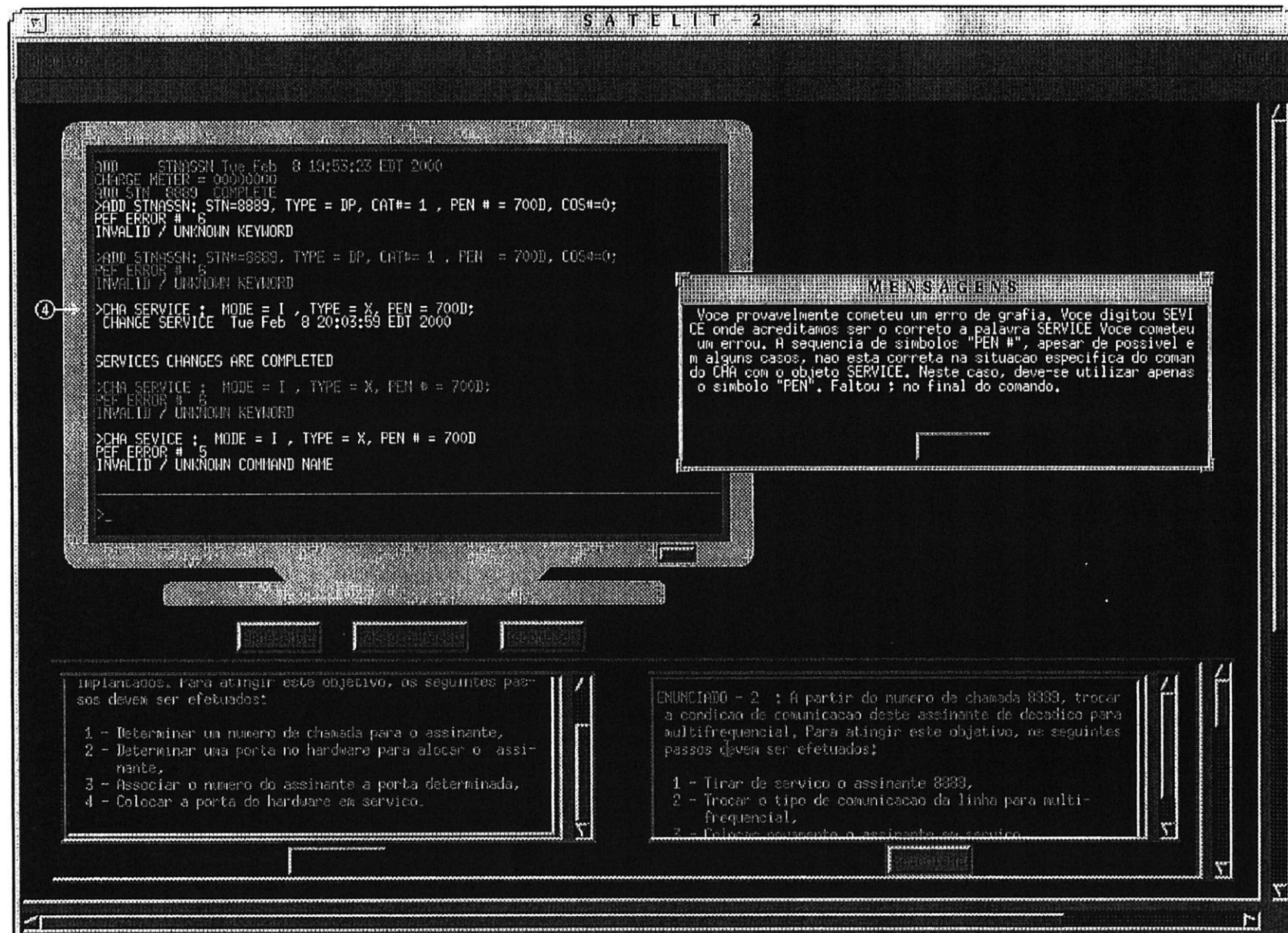
Este foi o único erro encontrado no comando, ou seja, está na categoria de erros singulares. É possível corrigir o erro (e esta correção está guardada no ambiente de ensino), mas optou-se por não se fazer a correção automática do erro. Na fase de análise da lógica de programação esta correção pode ser utilizada pelo ambiente para determinar se o comando está logicamente correto dentro do contexto.

Um pouco depois, na resolução do mesmo enunciado, outro erro ocorre por parte do aprendiz. O erro pode ser vizualizado na Figura 7 e demonstra a situação quando o simulador recebe o seguinte comando:

CHA SEVICE : MODE = I , TYPE = X PEN # = 700D

Após a passagem deste comando pelo MAES foram encontrados os erros de ortografia no objeto **SEVICE** onde foi possível aproximar para SERVICE, inclusão do símbolo **#** depois de PEN (sendo que neste contexto isso não é possível, apesar de ser em outros) e a falta do delimitador **;** no final do comando.

Todos esses erros são independentes entre si, ou seja, a ocorrência de um deles não implica no aparecimento de outros falsos erros. Aliás, mesmo que isso tenha ocorrido (por exemplo por não encontrar o objeto SEVICE e acumular outros erros por causa deste), o algoritmo de busca foi capaz de determinar a melhor escolha e corrigir os erros na ordem correta.

Figura 7 - **Snapshot do sistema SATELIT**

4. PROPOSTA DE APOIO À LÓGICA DE OPERAÇÃO NO SISTEMA SATELIT

Neste capítulo será descrito o trabalho executado no sistema SATELIT para o apoio ao operador na aquisição de perícia de programação. Como apresentado na introdução, por apoio à aquisição de perícia entenda-se a sua contribuição para a lógica de um programa (a seqüência de comandos a serem executados como parte da solução de um problema). Primeiro será mostrada a arquitetura do Modelo de Apoio ao Ensino de Lógica de operação (MAEL). Em seguida, são apresentadas as técnicas para o tratamento de erros de lógica de operação para que seja provida a instrução apropriada ao aprendiz.

4.1 ARQUITETURA DO MAEL

O MAEL é composto por 4 elementos como mostra a Figura 8:

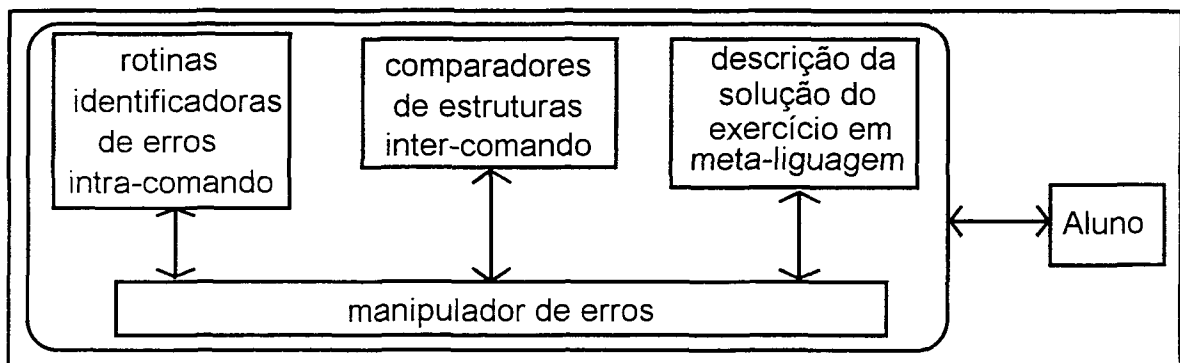



Figura 8

A seguir esses elementos são explicados com mais detalhes.

4.1.1 REPRESENTAÇÃO DA SOLUÇÃO DE UM EXERCÍCIO

Para que um sistema tutor seja capaz de dar apoio à lógica de programação é necessário que tenha um modelo interno prévio da solução do problema proposto ao operador. Esse modelo pode ser provido por um professor humano, por autoria

externa ao sistema tutor, ou então o próprio tutor pode ser capaz de "gerar" uma solução para o problema. Esta segunda possibilidade é muito mais rara na literatura de Sistemas Tutores Inteligentes pois implica na construção de sistemas que sejam capazes de resolver problemas no domínio específico e ainda ensinar a solução (MURRAY, 1999). Nos poucos casos onde ocorre essa integração entre o "resolvedor de problemas no domínio" e o "interpretador pedagógico", as publicações não apresentam fundamentos claros sobre a engenharia de construção do sistema.

Neste trabalho optou-se por dar linguagem e ferramentas para um autor humano apresentar a solução do problema já pronta. Em posse desse modelo de solução, é possível identificar se a combinação de passos executados pelo operador está correta ou não, por meio de um interpretador pedagógico genérico que também seja capaz de oferecer críticas ("*feedback*").ao aprendiz em boa parte dos casos onde este comete erros na solução.

Para descrever a solução do problema de uma maneira genérica e que o tutor pudesse entender, foi definida uma MEta-linguagem para a Descrição de Aspectos da Lógica de soluções de problemas (MEDAL). Usando esta meta-linguagem o instrutor humano define um conjunto razoável das variações de soluções de um exercício e elicita tais soluções de acordo com o modelo na linguagem objeto de operação.

4.1.1.1 A META-LINGUAGEM MEDAL

Como parte deste trabalho de Mestrado, foi definida a linguagem MEDAL (MEta-linguagem para a Descrição de Aspectos da Lógica de solução de um problema). Para tanto, o primeiro passo foi o de analisar um grande número de soluções e identificar quais situações de variação podem ocorrer na solução de forma invariavelmente correta. As principais estruturas genéricas identificadas são

as seguintes: opcionalidade, seqüência, ordem imposta ou não. Para essas situações foram definidas palavras-chave de controle. Essas palavras-chave compõem os comandos da meta-linguagem usada para descrever as variações nas soluções destes problemas. São elas:

- **alternativa exclusiva** -> somente um deles deve ser executado. É o caso onde dois caminhos mutuamente exclusivos são possíveis para solucionar um problema. Uma vez tomado um dos caminhos, é impossível voltar atrás e escolher outro caminho.
- **seqüência** -> a ordem é obrigatória. Dois (ou mais) comandos devem ser executados em uma determinada ordem. Por exemplo, deve-se primeiro abrir um arquivo para depois ler o que ele contém. Estes comandos executados em ordem diferente não terão o mesmo efeito.
- **opcional** -> pode ou não ser executado. Como o próprio nome diz, este comando é facultativo.
- **obrigatórios em qualquer ordem** -> todos devem ser executados, mas a ordem não importa. Aqui a ordem não importa, mas a obrigatoriedade de existência sim.

A Figura 9 mostra a representação gráfica destas palavras-chave. Em **AA**(a) está representada a alternativa exclusiva; em **AA**(b) a seqüência; em **AA**(c) opcional e em **AA**(d) os obrigatórios em qualquer ordem.

Acredita-se que apenas estas quatro palavras-chave são suficientes para representar o controle de qualquer solução para exercícios propostos. A princípio, foram feitos experimentos com exercícios destinados apenas para cadastrar e descadastrar assinantes, mudar o *status* de um assinante e gravar permanentemente os dados dos assinantes.

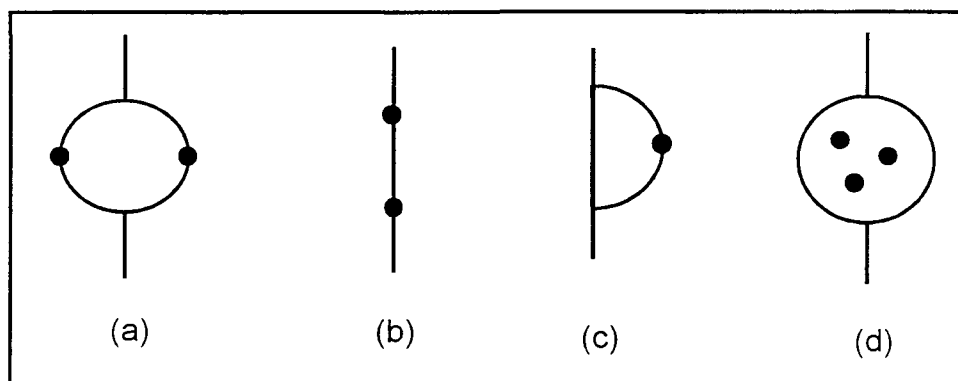



Figura 9 - **Representação gráfica das palavras-chave do MAEL**

O próximo passo foi escolher alguns enunciados de problemas que seriam tipicamente propostos e resolvidos durante o treinamento de operadores.

4.1.1.2 EXERCÍCIOS ANALISADOS

Para este trabalho foi escolhido um conjunto de enunciados de problemas dentre os mais comumentes utilizados pelos operadores de uma CDCT. Este conjunto é composto pelos seguintes elementos:

- cadastrar um assinante com variações de número de chamada e facilidades de serviços digitais dado o número da porta de hardware e o número de chamada do assinante;
- eliminar um assinante dado o seu número de chamada;
- trocar o status de um assinante de decádico para multi-freqüencial;
- trocar o status de um assinante de multi-freqüencial para decádico;
- implantar o serviço HOTLINE[1] para um assinante;
- implantar o serviço WARMLINE para um assinante;
- salvar os dados referentes a assinantes.

Para cada um destes exercícios foi definida uma solução utilizando a meta-linguagem descrita previamente. As soluções são definidas com base em um grafo

---

[1] Significa Linha Direta. Permite que um assinante origine uma chamada automaticamente, para um número pré-programado, apenas tirando o aparelho telefônico do gancho.

dirigido e acíclico, mais especificamente, uma árvore por nós denominada Árvore Para Prescrever a Lógica da solução de Exercícios (APPLE).

4.1.1.3 ÁRVORE DE REPRESENTAÇÃO DA SOLUÇÃO

A árvore APPLE é utilizada para representar a solução de um exercício proposto. Sua representação gráfica pode ser vista na Figura 10. Cada nó da árvore contém ou uma palavra-chave de controle (_OPCIONAL_, _SEQUENCIA_) ou uma estrutura com uma representação do comando a ser executado pelo operador. Esta estrutura, a princípio, tinha apenas o comando e uma sentença com o significado do comando. A sentença é usada para dar mensagens de erro ou acerto ao operador.

Mas estes campos não são suficientes para que o algoritmo manipulador de erros seja capaz de trabalhar. Assim, outros campos foram acrescentados à medida em que foram necessários. Os campos são:

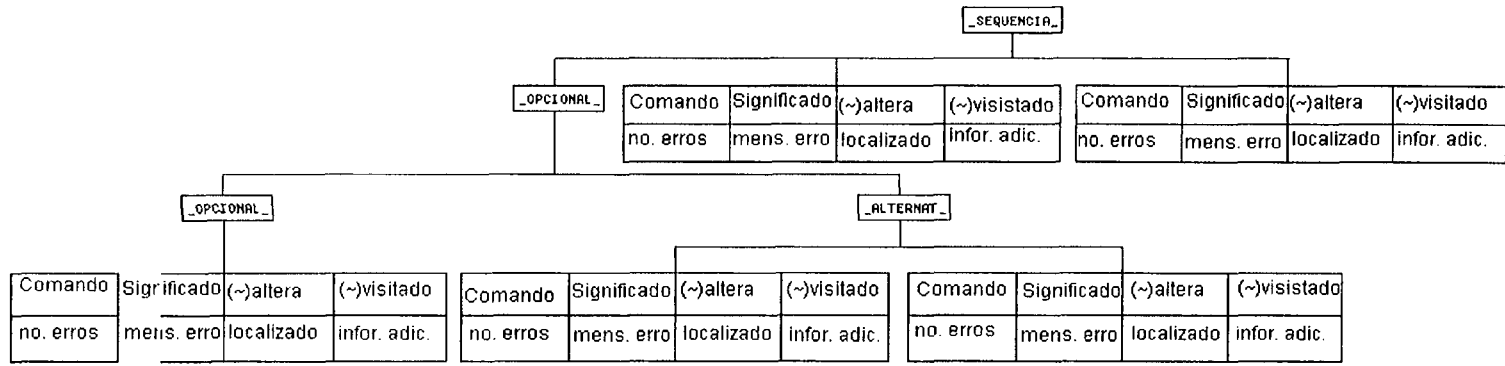
###• altera ou não -> um campo *booleano* dizendo se o comando altera o não o banco de dados do equipamento objeto. É necessário para determinar se a ocorrência deste comando pode ser considerado opcional ou se, em caso de erro, é necessário voltar a um estado anterior da base de dados do simulador;

- visitado ou não -> um campo *booleano* indicando se o nó já foi ou não visitado pelo algoritmo de acompanhamento dentro do escopo do exercício. É necessário porque a cada novo comando, a árvore é percorrida a partir da raiz. Utilizando este campo é possível determinar qual a relação do comando corrente em relação a todos os outros já visitados durante a execução da solução. Esta relação é então comparada com *templates* para determinar um erro ou acerto;
- número de erros -> este campo indica quantos erros este nó apresenta quando comparado ao comando do operador. Este campo é usado para

determinar quantos erros já foram encontrados no nó. Através deste parâmetro pode-se detectar qual o nó mais promissor. Também pode ser utilizado como critério de parada quando um ramo infrutífero é encontrado;

- mensagem de erro/acerto -> este campo contém uma mensagem para o aprendiz operador dizendo o que ele errou ou um comentário atestando o acerto. Este campo retorna ao aprendiz as mensagens acumuladas do nó. É a resposta do algoritmo para o usuário;
- localizado -> este campo demonstra se o nó já foi localizado na iteração corrente do algoritmo de tratamento de erros. Durante a execução do algoritmo é necessário verificar se um determinado nó já foi ou não visitado. Isso pode indicar uma repetição de comando, por exemplo.
- informações para o tratamento intra-comando -> este campo é um vetor com informações para o tratamento semântico do comando. Checagem de parâmetros, valores de parâmetros, por exemplo.

Com esses campos é possível determinar qual o cenário dentro de uma solução para um problema proposto. Uma vez definidos estes pontos, os possíveis erros foram classificados para que rotinas identificadoras pudessem ser implementadas.

Figura 10 - **Representação gráfica da árvore APPLE**

4.1.2 CLASSES IDENTIFICADORAS DE ERROS

A exemplo dos erros sintáticos, também foram criados identificadores de classes de erros de lógica. Estes identificadores recebem o comando semanticamente correto e o comando digitado pelo operador da central digital. As classes então procuram discrepâncias entre os dois comandos recebidos e as reportam como erros para o módulo de tratamento de erros horizontais.

Os identificadores foram definidos a partir de discussões empíricas sobre quais erros são passíveis de identificação. Entre as classes definidas estão as seguintes:

- alocação de número de assinante já utilizado, que identifica se o comando está tentando alocar um número já utilizado por outro assinante;
- alocação de porta de *hardware* já utilizada que identifica se o comando está tentando alocar uma porta já utilizada por outro assinante;
- parâmetro com valor válido porém incorreto, que identifica se o valor dado como parâmetro pelo operador corresponde ao valor especificado na solução dada pelo instrutor;
- objeto válido porém incorreto, que identifica se o objeto é válido para o comando, mas não é o correto de acordo com a solução especificada;
- não alterações de status no uso de um comando que altera o status da CDCT, que identifica que não houve alteração no status da central mesmo que um que deveria provocar alterações foi executado;

4.1.3 MÓDULO PARA O TRATAMENTO DE ERROS

Este módulo recebe o comando emitido pelo aprendiz-operador e faz a detecção e possível correção de erros semânticos (intra-comando) e de lógica

(inter-comando). Para fazer o diagnóstico de erros este módulo interage com os identificadores de classes de erros e a descrição do exercício corrente.

Na próxima seção este módulo será descrito com maiores detalhes.

4.2 TRATAMENTO DE ERROS

Nesta seção será visto o que foi definido para o tratamento de erros semânticos e de lógica em comandos. Os erros de lógica tratam de erros de estratégia na programação enquanto que os erros semânticos lidam com problemas de contexto de um único comando.

Logo de início foram detectados dois "eixos" diferentes onde um comando poderia estar errado. A Figura 11 mostra estes eixos.

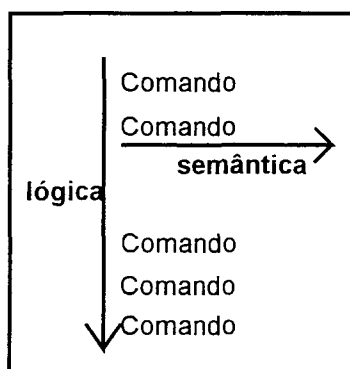



Figura 11 - **Eixos de detecção de erros para apoio lógico**

O eixo vertical mostra os erros de lógica (inter-comandos). Estes erros são ocasionados por erro de estratégia do programador. A questão principal que fundamenta este tipo de erro chama-se ordem de entrada de comandos.

Os erros intra-comando ocorrem ao longo do eixo horizontal, portanto ao longo de um único comando. Este tipo de erro é um pouco mais difícil de detectar. Eles tratam principalmente de erros dinâmicos, que só são identificáveis durante a

execução. A abordagem para detectar e corrigir estes dois tipos de erros é diferente uma da outra. E por isso serão descritas nos próximos tópicos separadamente.

4.2.1 TRATAMENTO DE ERROS INTER-COMANDOS

Os erros lógicos, como descrito anteriormente, são aqueles que ocorrem ao longo do eixo vertical quando da análise de um programa. Praticamente todos os erros que ocorrem ao longo do eixo vertical são derivados de: (1)comandos programados em ordem incorreta, (2)falta de um comando, (3)inclusão de comando indevido.

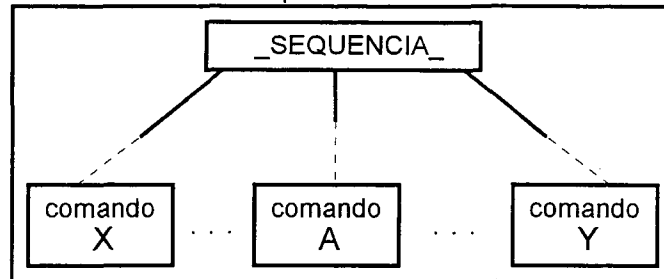
Um exemplo de erro lógico é tentar consultar o valor de uma variável sem que esta tenha sido alocada ou que anteriormente tenha recebido um valor significativo. Ou, mais especificamente no caso de uma CDCT, tentar acessar um assinante ainda não cadastrado, por exemplo.

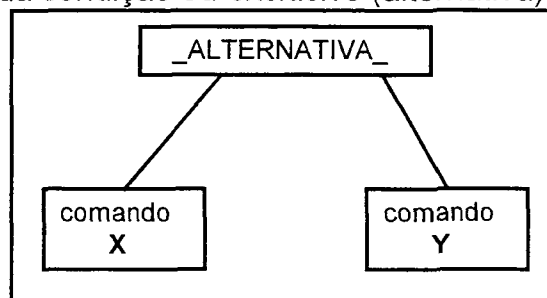
Para detectar estes erros será adotada futuramente a técnica de *model tracing* (ANDERSON, 1984). Esta técnica, anteriormente descrita na seção 2.2.2, consiste em acompanhar os passos do programador aprendiz comparando-os a variações de uma solução pré-existente. Esta solução pré-existente neste caso é a descrição do exercício. A descrição, como visto na seção 4.1.1, é feita na forma da árvore APPLE. Portanto, a procura de uma folha que "case" (ou corresponda) com o comando do programador é um algoritmo que percorre uma árvore em busca de um nó.

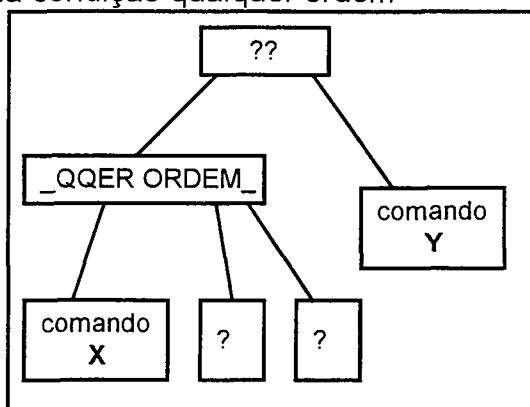
Este algoritmo percorre a árvore APPLE numa busca em profundidade até encontrar um nó folha que corresponda ao comando do operador. Para detectar se os nós são correspondentes compara-se o comando do operador e o comando contido no nó.

Depois de encontrado o nó na árvore APPLE, é necessário analisar se este nó está numa seqüência viável quando comparado ao último nó encontrado (último

comando executado anteriormente). Para fazer este teste de viabilidade foram criados algumas "máscaras" (*templates)* de diferentes situações de erro que uma árvore APPLE pode apresentar. Dentre estes *templates* podem-se destacar os seguintes:

- falta de comando em seqüência

o comando X foi o último comando executado com sucesso e Y é o comando atual. As reticências indicam que o comando X está numa subárvore de _SEQUENCIA_, sem necessariamente ser seu filho direto. O mesmo vale para os nós A e Y. As reticências entre os nós X - A - Y indicam que tem de haver ao menos um nó filho entre X e Y para que este erro seja detectado.

- violação da condição ou-exclusivo (alternativa)

o comando X foi o último comando executado com sucesso e o Y é o comando corrente. Ambos estando abaixo de um nó _ALTERNATIVA_ indica que eles são mutuamente exclusivos. Então, para que o erro seja detectado basta que haja um nó já visitado abaixo do nó _ALTERNATIVA_.

- violação da condição qualquer ordem

o comando X foi o último comando executado com sucesso e Y é o comando corrente. O nó _QQER ORDEM_ implica que todos os seus filhos sejam executados (não importando a ordem) antes que outro ramo "irmão" dele seja executado. Portanto, quando o nó corrente está fora desta sub-árvore, o erro é detectado.

4.2.2 TRATAMENTO DE ERROS INTRA-COMANDO

Os erros intra-comando são aqueles que ocorrem ao longo do eixo horizontal ao longo de um único comando. Este tipo de erro trata principalmente de "contexto" local. Por isso, na maioria das vezes, só pode ser verificado durante a execução do programa. Alguns exemplos destes erros são tentar cadastrar um assinante que já existe ou disponibilizar um serviço para o assinante sem que ele preencha os pré-requisitos necessários.

Para verificar esses casos é necessário fazer uma consulta ao banco de dados da central digital. Uma vez que este banco de dados é altamente dinâmico, não é possível saber com antecedência o estado do mesmo, daí a necessidade de verificação durante a execução do programa.

Para a detecção deste tipo de erro será adotada uma abordagem parecida com a utilizada para a detecção de erros sintáticos. Foram definidas na seção 4.1.2 algumas classes de erros semânticos e suas respectivas rotinas identificadoras de erros às quais o comando corrente é submetido para verificação Estas rotinas

encontram-se implementadas, mas ainda não integradas ao sistema. O comando detectado como possível candidato pelo algoritmo de busca de erros lógicos é submetido a todas as classes em ordem seqüencial e acumula as mensagens dos erros de lógica que porventura tenham ocorrido.

Ao contrário do que ocorre com o processo de detecção de erros sintáticos (Capítulo 3), aqui não importa muito a ordem em que o comando é submetido a essas classes. Apesar disso, para melhorar o entendimento por parte do aprendiz, as classes com maior probabilidade de ocorrer são as primeiras verificadas. Esta probabilidade será determinada de duas formas empíricas: (a) por meio da elicitação de conhecimento de tutores humanos que sejam especialistas em estimativas de ocorrência de erros de aprendizes durante o treinamento; (b) por meio de testes estatísticos diretamente efetuados nos arquivos contendo a cópia das atividades de treinamento.

5 CONCLUSÃO

Neste capítulo será feita uma breve explanação sobre as soluções propostas por este trabalho. As vantagens e desvantagens serão expostas e analisadas, mostrando o porquê de algumas decisões tomadas.

5.1 APLICABILIDADE PARA OUTRAS LINGUAGENS DE OPERAÇÃO DE CENTRAIS DIGITAIS

Desde o início, o propósito deste trabalho era não só limitar-se a apoiar o ensino de operação da CDCT SPX2000, mas permitir que a solução aqui proposta pudesse ser apresentada para outras centrais. Portanto, o que foi idealizado aqui levou esse fato em consideração.

O simulador trabalha sobre uma representação em meta-linguagem, o que faz com que ele fique independente da linguagem-objeto ensinada. Tanto o módulo de tratamento de erros de sintaxe quanto os erros de lógica fazem uso de meta-linguagens. Em nenhum momento a linguagem da CDCT SPX2000 é utilizada em sua forma original.

Alguns estudos preliminares já foram feitos para descrever a linguagem de outra CDCT (por exemplo a EWSD) utilizando as mesmas meta-linguagens. Praticamente nenhum grande obstáculo foi encontrado. O que sugere que é possível utilizar a solução proposta para outras centrais. Esta característica de reusabilidade é o que faz uma "*shell*" ser constituída de mecanismos tão complexos.

Uma vez tendo a "*shell*" pronta, os custos para produzir outro produto que a utilize são minimizados. Falando em custos, há outra facilidade proporcionada por este trabalho que ajuda a reduzir os custos: o uso de um simulador.

O treinamento tradicional para a operação de uma CDCT requer que uma central seja alocada para o mesmo. Além do agravante de manter uma central real parada para efeitos de treinamento, ainda é necessário que os alunos se desloquem ao centro de treinamento. Ou seja, o treinamento remoto é impossível. Utilizando um sistema computadorizado dotado de simulador isso não ocorre. O treinamento à distância pode se tornar uma realidade, ou então o curso vai se deslocar até os aprendizes, ao contrário do que ocorre hoje.

Além do fato de cada aluno ter uma central totalmente para si, ele também tem um professor dedicado a ele. Não que a figura do tutor humano seja plenamente substituída pelo sistema tutorial. Mas o sistema pode se ocupar das tarefas mais simples, como o tratamento da sintaxe de um comando, por exemplo, e deixar para o tutor humano as tarefas mais complexas, tais como a melhor compreensão dos aspectos de um enunciado. Dessa forma a perícia do tutor humano seria melhor aproveitada durante um curso de treinamento.

5.2 LIMITAÇÕES DA FERRAMENTA E LINGUAGEM

Apesar de todas as vantagens apresentadas no item anterior, é necessário ressaltar alguns pontos quanto a possíveis limitações encontradas. Até o presente momento ainda é necessário implementar partes da base de conhecimento utilizando o mesmo software usado para construir o sistema SATELIT (DIRENE, 1997). Este fato gera uma indesejada dependência entre o sistema e a linguagem a ser ensinada. A saber, o sistema SATELIT, bem como o módulo de apoio sintático foi implementado em POPLOG (BARRET; RAMSAY; SLOMAN, 1985).

Foi detectado que o simulador também deve sofrer ajustes para trabalhar com uma nova linguagem. Isso se deve ao fato de que as centrais operam de

maneira diferente para executar uma determinada tarefa. Além disso, a meta-linguagem MELON não guarda em si elementos da semântica dos comandos. Essa parte é implementada em separado, no simulador, o que torna o código da "*shell*" parcialmente dependente do domínio.

Já a meta-linguagem MEDAL, utilizada para dar apoio à lógica, prevê elementos para guardar informações de semântica. Ainda não foi analisado quais os impactos que ela deve sofrer quando da implementação de uma nova linguagem de operação. Mas provavelmente as modificações sejam um pouco menores que na MELON e/ou no simulador.

De maneira análoga, o MAES também deve sofrer uma adaptação maior que o MAEL. Da maneira como está hoje, o MAES preocupa-se apenas com a sintaxe de um comando. O MAEL trabalha com uns poucos elementos de semântica; seu maior foco é na lógica de programação.

Esses pontos mostram que é necessário estudar uma maneira de contemplar a representação semântica, mesmo que de forma rudimentar, na própria meta-linguagem que descreve a linguagem-objeto ou mesmo numa estrutura à parte. Dessa forma o simulador da central digital sofreria menos impactos quando da passagem para outra linguagem.

Claramente alguns pontos são inerentes a cada central, forçando que o simulador sempre tenha que sofrer alguma adaptação. O objetivo é tentar minimizar o tamanho desta adaptação para que a reusabilidade da "*shell*" e do sistema como um todo seja elevada, visando o máximo.

5.3 CONSIDERAÇÕES FINAIS

Foram mostradas no presente trabalho a implementação e a proposta de dois módulos que estão embutidos em um STI para apoiar o ensino da operação e manutenção de uma CDCT. Utilizando esses módulos espera-se proporcionar diminuição do tempo de treinamento, além do aumento do potencial de diagnóstico de erros por parte do aluno. A utilização também facilita o treinamento à distância, que não é praticado atualmente por inviabilidade técnica.

Pelo fato da CDCT ser operada através de uma linguagem formal de programação, foram citados problemas já detectados nesta área por outros pesquisadores. De modo geral, verificou-se que o aluno precisa dominar os princípios e a perícia para ser considerado um especialista.

Por isso os módulos aqui descritos visam justamente atacar estes dois problemas. Na literatura verificou-se que geralmente os sistemas tratam de ambos os tópicos de maneira conjunta. Aqui eles foram separados em dois módulos distintos, com soluções separadas.

Cada um desses dois módulos foi então aberto e explicado em detalhes. No módulo de apoio à sintaxe, já implementado, foram mostrados exemplos práticos de aplicação. Já o módulo de apoio à lógica foi somente proposto e especificado neste trabalho.

Sua implementação está no escopo de trabalhos futuros. Outras propostas para o futuro incluem a adaptação do sistema SATELIT, do qual este trabalho faz parte, para outra CDCT. Também é interessante implementar uma ferramenta de autoria para que os instrutores sejam capazes de criar novos elementos para a galeria de exercícios apresentada pelo sistema ao aluno.

Dependendo do grau de independência que se consiga atingir, será possível implementar uma ferramenta de autoria para que o especialista descreva a

linguagem de operação de outra CDCT. Quando tal fato ocorrer, os objetivos deste trabalho, que são: diminuir o tempo de treinamento, diminuir os custos, dar apoio ao ensino de princípios e perícia, terão sido plenamente alcançados.

ANEXO 1

A LINGUAGEM DA CDCT SPX2000 DESCRITA EM MELON

Os símbolos usados na descrição da linguagem apresentam os seguintes significados:

- META-SÍMBOLOS:

=> (produz)

| (ou-exclusivo)

{ } (repetição indefinida)

[ ] (opcional - internamente a cada comando ou parâmetro)

- SÍMBOLOS NÃO TERMINAIS ESPECIAIS (convenção):

$ (dígito - 0 a 9)

@ (hexadecimal - 0 a F)

~ (espaço em branco)

A seguir estão os comandos descritos em MELON:

ADD STNASSN : STN # '=' _valStn_, TYPE '=' _valType_, CAT# '=' _valCat_, PEN # '=' _valPen_, COS # _valCos_ [, ORIG '=' _valOrig_] [,EFLG '=' _valEflg_] [,RSTR '=' _valRstr_] [, RSCT '=' _valRsct_] [, WARM '=' _valWarm_] [,IDX # '=' _valIdx_] ;

valStn => $$$$

valType => DP | DTMF | HOTLINE

valCat => 1 | 2 | 3 | 4 | 5 | 6 | 7

valPen => $$$@

valCos => $ | $$

```
_valOrig_        => $
_valRstr_        => $ | $$
_valRsct_        => $ | $$
_valWarm_        => $ | $$
_valIdx_         => $ | $$
```

CHA STNASSN : OSTN '=' _valOstn_ [, STN # '=' _valStn_] [, CHBU '=' _valChbu_] [, CHDN '=' _valChdn_] [, TYPE '=' _valType_] [, CAT # '=' _valCat_] [, ORIG '=' _valOrig_] [, ORGC '=' _valOrgc_] [, COS # '=' _valCos_] [, ERST '=' _valErst_] [, DRST '=' _valDrst_] [, ERSC '=' _valErsc_] [, DRSC '=' _valDrsc_] [, EWRM '=' _valEwrm_] [, DWRM '=' _valDwrm_] [, EHID '=' _valEhid_] [, DHID '=' _valDhid_] ;

```
_valOstn_        => $$$$
_valChbu_        => Y | N
_valChdn_        => Y | N
_valOrgc_        => Y | N
_valErst_        => $ | $$
_valDrst_        => Y | N
_valErsc_        => $ | $$
_valDrsc_        => $ | $$
_valEwrm_        => $ | $$
_valDwrm_        => Y | N
_valEhid_        => $ | $$
_valDhid_        => Y | N
```

DEL STNASSN : STN # '=' _valStn_ [, CHBU '=' _valChbu] [, CHDN '=' _valChdn_] ;

DIS STNASSN [: STN # '=' _valStn_] ;

HEL STNASSN ;

CHA SERVICE : MODE '=' _valMode_ [, PENC '=' _valPenc_] [, DISC '=' _valDisc_] [, TYPE '=' _valType_] [, PEN '=' _valPen_] [, DIS '=' _valDis_] [, CHNU '=' _valChnu_] [, DIST '=' _valDist_] [, DINU '=' _valDinu_] [, EXTN '=' _valExtn_] ;

valMode => C | I | M | N | D

valPenC => $$$

valDisC => Y | N

valCSType => P | D | D32 | X

valDis => Y | N

valChnu => $ | $$

valDist => Y | N

valDinu => Y | N

valExtn => $$$$

DIS SERVICE : PEN '=' _valPen_ [, MODE '=' _valModeC_] [, EXTN '=' _valExtn_] ;

_valModeC => ALL | IN | MTCE | OUT | OUTAUTO| OUTAUTOPEND | OUTCRAFT | OUTCRAFTCAMP | OUTCRAFTPEND

HEL SERVICE ;

DIS DEVDATA : PEN '=' _valPen_ [, LST '=' _valLst_] ;

valLst => $$$H | $$$T

HEL DEVDATA ;

CHA HOTLINE : IDX # '=' _valIdx_ [, DGTS '=' _valDgts_] ;

valDgts => $$$$$$$$$$$$$$$$ | $$$$$$$$$$$$$$$ | $$$$$$$$$$$$$$ | $$$$$$$$$$$$$ | $$$$$$$$$$$$ | $$$$$$$$$$$ | $$$$$$$$$$ | $$$$$$$$$ | $$$$$$$$ | $$$$$$$ | $$$$$$ | $$$$$ | $$$$

DIS HOTLINE : HOTTYPE '=' _valHotType_ [, IDX # '=' _valIdx_] ;

valHotType => D | I

HEL HOTLINE ;

ADD STNCOS : COS # '=' _valCos_ [, CLMS '=' _valClms_] ;

valClms => CWTERM | DND | FWDALL | FWDBUSY | FWDNOANS | FDWTOSECR | LASTNO | MALICID | SCIND | SPLIT | WARMLINE | XFERAUTO | CONFSTN (este serviço não é permitido p/ o SPX2000 configuração 256 assinantes)

CHA STNCOS : COS # '=' _valCos_ [, DCLM '=' _valClms_] [, ECLM '=' _valClms_] ;

DEL STCOS : COS # '=' _valCos_ ;

DIS STCOS : COS # '=' _valCos_ [, ALL? '=' _valAll_] ;

valAll => Y | N

HEL STNCOS ;

CHA TIMEDATE : TIME '=' _valTime_ , DATE '=' _valDate_ ;

valTime => $$ $$ $$

valDate => $$ $$ $$

DISTIMEDATE ;

HEL TIMEDATE ;

DIS ALMDATA : ALMTYPE '=' _valAlmType_ ;

valAlmType => NEW | ALL

HEL CUSTDATA ;

SAV CUSTDATA [: SYST '=' _valSyst_] ;

valSyst => ACTIVE | STANDBY | BOTH

ANEXO 2

DESCRIÇÃO DE TODAS AS CLASSES DE ERROS SINTÁTICOS E EXEMPLOS DE OCORRÊNCIAS

1 - ERRO ORTOGRÁFICO

- Exemplo de comando incorreto:

ADD STNASN: STN#=8889, TYPE = DP, CAT#= 1 , PEN # = 700D, COS#=0;

- Rotina de correção

Corrige_Erro_Ortografico(...) -> Ocorreu_Erro_ -> Msg_Erro_

2 - SUBSTITUIÇÃO DE SÍMBOLOS DEVIDO À SEMELHANÇA VISUAL

- Descrição:

Substituição indevida de símbolo (não alfanumérico) por outro símbolo (não alfanumérico) existente na linguagem, devido à semelhança visual entre eles. Por exemplo: [":" ";" "|"], ou ["/" "\"], ou ["0" "O"] etc.

- Exemplo de comando incorreto:

ADD STNASSN ; TYPE = DP, CAT#= 1 , STN#=8889,PEN # = 700D, COS#=0;

ADD STNASSN | TYPE = DP, CAT#= 1 , STN#=8889,PEN # = 700D, COS#=0;

- Rotina de correção:

Corrige_Erro_Subst_Simb_Semelh_Visual(...) -> Ocorreu_Erro_ -> Msg_Erro_

3 - ESQUECIMENTO DE SÍMBOLOS COMUMENTE USADOS COMO SEPARADORES

- Descrição:

Erro por esquecimento de símbolos (não alfanuméricos) que, em diversas linguagens, são freqüentemente utilizados como separadores. Por exemplo: "," ";" ":"

- Exemplo de comando incorreto:

ADD STNASSN: STN#=8889, TYPE = DP CAT#= 1 , PEN # = 700D, COS#=0;

- Rotina de correção:

Corrige_Erro_Esquec_Simb_Freq_Separad(..) -> Ocorreu_Erro_ -> Msg_Erro_

4 - ESQUECIMENTO SISTEMÁTICO DE SÍMBOLOS COMUMENTE USADOS COMO SEPARADORES

- Descrição:

Erro por esquecimento de todas as ocorrências de um mesmo símbolo (não alfanumérico) que, em diversas linguagens, é freqüentemente utilizado como separador. Por exemplo: "," ";" ":"

- Exemplo de comando incorreto:

ADD STNASSN: STN#=8889 TYPE = DP CAT#= 1 PEN # = 700D COS#=0;

- Rotina de correção

Corrige_Erro_Esquec_Sistemat_Simb_Freq_Separad(.) -> Ocorreu_Erro_ -> Msg_Erro_

5 - ESQUECIMENTO DE SÍMBOLOS EM SEQÜÊNCIA COMBINADA

- Descrição:

Erro por esquecimento de um símbolo (não alfanumérico) que, em parte ou na totalidade dos casos, aparece combinado, e em seqüência, com outro(s) símbolo(s)

(alfanumérico(s) ou não) de categoria sintática semelhante (por exemplo, um parâmetro de comando). Por exemplo: "PEN" "#" ou "CAT" "#"

- Exemplo de comando incorreto:

ADD STNASSN: STN=8889, TYPE = DP, CAT#= 1 , PEN# = 700D, COS#=0;

- Rotina de correção:

Corrige_Erro_Esquec_Simb_Sequen_Combinada(..) -> Ocorreu_Erro_ -> Msg_Erro_

6 - INCLUSÃO INDEVIDA DE SÍMBOLOS EM SEQÜÊNCIA NÃO COMBINADA

- Descrição:

Erro por inclusão indevida de um símbolo (não alfanumérico) que, em parte ou na totalidade dos casos de outro(s) símbolo(s) (alfanumérico(s) ou não) de categoria sintática semelhante, aparece combinado, e em seqüência, com tal(is) símbolo(s) (por exemplo, um parâmetro de comando). Por exemplo: "TYPE" "#"

- Exemplo de comando incorreto:

ADD STNASSN: STN#=8889, TYPE# = DP, CAT#= 1 , PEN # = 700D, COS#=0;

- Rotina de correção:

Corrige_Erro_Inclus_Indevida_Simb_Sequen_Combinada(..) -> Ocorreu_Erro_ -> Msg_Erro_

7 - INVERSÃO (2 A 2) DE VALORES EXCLUSIVOS DE PARÂMETROS

- Descrição:

Erro por inversão de valores de parâmetros, onde ambos os valores envolvidos são exclusivos (2 a 2) na comparação do universo de valores possíveis da definição dos parâmetros.

- Exemplo de comando incorreto:

ADD STNASSN: STN#=8889, TYPE = 1, CAT#= DP , PEN # = 700F, COS#=0;

(Quando "deveria" ser, por exemplo:

ADD STNASSN: STN#=8889, TYPE = DP, CAT#= 1 , PEN # = 700F, COS#=0;)

- Rotina de correção:

Corrige_Erro_Inversao_Val_Param(.) -> Ocorreu_Erro_ -> Msg_Erro_

8 - ERROS DIVERSOS EM VALORES DE PARÂMETROS

- Descrição:

Identificação de erros variados em valores de parâmetros, tais como: tamanho de caracteres/dígitos, escopo dos valores, e outros.

- Exemplo de comandos incorretos:

ADD STNASSN: STN#=889, TYPE = DP, CAT#= 1 , PEN # = 700D, COS#=0;

ADD STNASSN: STN#=86789, TYPE = DP, CAT#= 1 , PEN # = 700D, COS#=0;

9 - ERROS DE CARACTERES INVÁLIDOS EM VALORES DE PARÂMETROS

- Descrição:

Identificação de erros específicos de caracteres inválidos em valores de parâmetros, tais como: letra fora da faixa de símbolo hexadecimal, troca de "O" por "0" (e vice-versa), e outros.

- Exemplo de comando incorreto:

ADD STNASSN: STN#=8889, TYPE = DP, CAT#= 1 , PEN # = 700K, COS#=0;

10 - INCLUSÃO DE ESPAÇOS EM BRANCO ONDE NÃO PODE HAVER

- Descrição:

Inclusão de espaços em branco onde a sintaxe não permite, mas que ainda podem ser identificados por concatenação e validação do símbolo seguinte ao espaço com o símbolo anterior ao espaço.

- Exemplo de comandos incorretos:

ADD STNASSN: STN#=8 889, TYPE = DP, CAT#= 1 , PEN # = 700D, COS#=0;

ADD STNASSN: STN#=8889, TYPE = D P, CAT#= 1 , PEN # = 700D, COS#=0;

ADD STN ASSN: STN#=8889, TYPE = DP, CAT#= 1 , PEN # = 700D, COS#=0;

- Rotina de correção:

Corrige_Erro_Inclusao_Espacos_Branco(...) -> Ocorreu_Erro_ -> Msg_Erro_ ;

11 - SUBSTITUIÇÃO SISTEMÁTICA DE SÍMBOLOS SEMELHANTES

- Descrição:

Erro por substituição sistemática de símbolo (não alfanumérico) por outro símbolo (não alfanumérico) existente na linguagem, devido à semelhança visual entre eles. Por exemplo: [":" ";" "|"], ou ["/" "\"], ou ["0" "O"], ou ["1" "l" "|"] etc. (muitas vezes separadores).

- Exemplo de comando incorreto:

ADD STNASSN: STN#=700D; TYPE = DP; CAT#= 1 ; PEN # = 8889; COS#=0;

- Rotina para correção:

Corrige_Subst_Sistemat_Simb_Semelh_Visual(..) -> Ocorreu_Erro_ -> Msg_Erro_

12 - CONTRAÇÃO DE SÍMBOLOS

- Descrição:

Não inclusão de espaços em branco entre dois ou mais símbolos (ou termos) válidos da linguagem. Pode ser tentado **antes** de uma rejeição final por aproximação ortográfica sem sucesso.

- Exemplo de comando incorreto:

ADDSTNASSN: STN#=8889, TYPE = DP, CAT#= 1 , PEN # = 700D, COS#=0;

- Rotina de correção:

Corrige_Erro_Contracao_Simb(...) -> Ocorreu_Erro_... -> Msg_Erro_... ;

13 - INCOMPATIBILIDADE DE OBJETO COM RELAÇÃO AO COMANDO

- Descrição:

Uso de termo (alfanumérico) incompatível (ver Manual de O&M - pág. 2.5.5 parte 2) com o comando corrente, onde um outro termo deverá ser aconselhado para substituição.

- Exemplo de comando incorreto:

ADD ALMDATA: STN#=8889, TYPE = DP, CAT#= 1 , PEN # = 700D, COS#=0;

14- USO DE PARÂMETRO DE OUTRO COMANDO

- Descrição:

Uso de parâmetro de outro comando mas que não se aplica ao comando corrente.

- Exemplo de comando incorreto:

CHA SERVICE: MODE = I , STN#=8889, TYPE = X, PEN = 700D;

^

15 - REPETIÇÃO DE PARÂMETRO NO COMANDO

- Descrição:

Digitação repetida de um dado parâmetro, mantendo ou não seus valores idênticos.

- Exemplo de comando incorreto:

ADD STNASSN: STN#=8889, TYPE=DP, CAT#= 1, STN#=8889, PEN # = 700D, COS#=0;

- Rotina de correção

Corrige_Erro_Repet_Param_Coman(...) -> Ocorreu_Erro_... -> Msg_Erro_... ;

16 - USO DE PARÂMETROS ONDE NÃO HA'

- Descrição:

Tentativa de fornecer parâmetros a um comando combinado com termos(s) que, sem tais parâmetros, já formariam um comando completo.

- Exemplo de comando incorreto:

DIS ALMDATA: PEN # = 700D, COS#=0;

17 - FALTA DE DELIMITADOR DE FINAL DE COMANDO

- Descrição:

Caso seja sempre o mesmo, identificar a falta do delimitador de final de comando. Este deve ser o primeiro, ou um dos primeiros testes a serem aplicados no comando do aprendiz.

- Exemplo de comandos incorretos:

ADD STNASSN: STN#=8889, TYPE = DP, CAT#= 1 , PEN # = 700D, COS#=0

ADD STNASSN: STN#=8889, TYPE = DP, CAT#= 1 , PEN # = 700D, COS#=0,

- Rotina de correção:

Corrige_Erro_Falta_Delimit_Final_Comando(...) -> Ocorreu_Erro_ -> Msg_Erro_

18 - EXCLUSÃO INDEVIDA DE SÍMBOLOS EM SEQÜÊNCIA COMBINADA ONDE ORA APARECE - ORA NÃO APARECE.

- Descrição:

Dependendo do comando, um dado termo/parâmetro vem seguido de um dado símbolo/termo específico. Em outros comandos, o mesmo parâmetro não vem seguido do referido termo específico (Por exemplo: "PEN" seguido de "#"). Este erro ocorre por *NÃO* inclusão de um símbolo (alfanumérico(s) ou não), que hora aparece, hora não aparece, em seqüência combinada a um termo/parâmetro (comparar com 5.7 e 5.8).

- Exemplo de comando incorreto:

ADD STNASSN: STN#=8889, TYPE = DP, CAT#= 1 , PEN = 700D, COS#=0;

- Rotina de correção:

Corrige_Erro_Exclu_Simb_Sequen_Combinada(...) -> Ocorreu_Erro_ -> Msg_Erro_

19 - INCLUSÃO INDEVIDA DE SÍMBOLOS EM SEQÜÊNCIA COMBINADA ONDE ORA APARECE - ORA NÃO APARECE.

- Descrição:

Dependendo do comando, um dado termo/parâmetro vem seguido de um dado símbolo/termo específico. Em outros comandos, o mesmo parâmetro não vem seguido do referido termo específico (Por exemplo: "PEN" seguido de "#"). Este erro ocorre por inclusão *INDEVIDA* de um símbolo (alfanumérico(s) ou não), que ora aparece, ora não aparece, em seqüência combinada a um termo/parâmetro (comparar com 5.7 e 5.8).

- Exemplo de comando incorreto:

CHA SERVICE: MODE = I , STN#=8889, TYPE = X, PEN # = 700D;

*** quando deveria ser: ***

CHA SERVICE: MODE = I , STN#=8889, TYPE = X, PEN = 700D;

- Rotina de correção:

Corrige_Erro_Inclu_Indev_Simb_Seq_Comb_Nao_Comb(...) -> Ocorreu_Erro_ -> Msg_Erro_

20 - INVERSÃO DA ORDEM DE SÍMBOLOS/TERMOS CONSECUTIVOS

- Descrição:

Inversão da ordem em que dois Símbolos/Termos aparecem em um comando.

- Exemplo de comando incorreto:

ADD STNASSN: STN#=8889, TYPE = DP, CAT#= 1 , # PEN = 700D, COS#=0;

- Rotina de correção:

Corrige_Erro_Inversao_Ordem_Simbs(...) -> Ocorreu_Erro_ -> Msg_Erro_

21 - USO DE NOME DE COMANDO COMO PARÂMETRO

- Descrição:

Substituição indevida de um parâmetro por um nome de comando existente na linguagem, devido à semelhança de nomes ou outra razão qualquer.

- Exemplo de comando incorreto:

ADD STNASSN: TYPE = DP, CAT#= 1 , STN#=8889,PEN # = 700D, DEL #=0;

22 - USO DE NOME DE COMANDO COMO VALOR DE PARÂMETRO

- Descrição:

Substituição indevida de um valor de parâmetro por um nome de comando existente na linguagem, devido à semelhança de nomes ou outra razão qualquer.

- Exemplo de comando incorreto:

ADD STNASSN: TYPE = CHA, CAT#= 1 , STN#=8889,PEN # = 700D, COS#=0;

23- USO DE NOME DE COMANDO COMO OBJETO

- Descrição:

Substituição indevida de um objeto por um nome de comando existente na linguagem, devido à semelhança de nomes ou outra razão qualquer.

- Exemplo de comando incorreto:

ADD SAVE: TYPE = DP, CAT#= 1 , STN#=8889,PEN # = 700D, COS#=0;

24 - USO DE NOME DE OBJETO COMO VALOR DE PARÂMETRO

- Descrição:

Substituição indevida de um valor de por um nome de objeto existente na linguagem, devido à semelhança de nomes ou outra razão qualquer.

- Exemplo de comando incorreto:

ADD STNASSN: TYPE = DP, CAT#= ALMDATA, STN#=8889,PEN # = 700D, COS#=0;

25 - USO DE NOME DE OBJETO COMO PARÂMETRO

- Descrição:

Substituição indevida de um parâmetro por um nome de objeto existente na linguagem, devido à semelhança de nomes ou outra razão qualquer.

- Exemplo de comando incorreto:

ADD STNASSN: TYPE = DP, CAT#= 1 , STN#=8889,PEN # = 700D, CUSTDATA#=0;

26 - USO DE NOME DE OBJETO COMO COMANDO

- Descrição:

Substituição indevida de um comando por um nome de objeto existente na linguagem, devido à semelhança de nomes ou outra razão qualquer.

- Exemplo de comando incorreto:

STNASSN SERVICE: MODE = I , STN#=8889, TYPE = X, PEN = 700D;

27 - USO DE NOME DE PARÂMETRO COMO COMANDO

- Descrição:

Substituição indevida de um comando por um nome de parâmetro existente na linguagem, devido à semelhança de nomes ou outra razão qualquer.

- Exemplo de comando incorreto:

STN STNASSN: TYPE = DP, CAT#= 1 , STN#=8889,PEN # = 700D, COS#=0;

28 - USO DE NOME DE PARÂMETRO COMO VALOR DE PARÂMETRO

- Descrição:

Substituição indevida de um valor de parâmetro por um nome de parâmetro existente na linguagem, devido à semelhança de nomes ou outra razão qualquer.

- Exemplo de comando incorreto:

ADD STNASSN: TYPE = DISC, CAT#= 1 , STN#=8889,PEN # = 700D, COS#=0;

29 - USO DE NOME DE PARÂMETRO COMO OBJETO

- Descrição:

Substituição indevida de um objeto por um nome de parâmetro existente na linguagem, devido à semelhança de nomes ou outra razão qualquer.

- Exemplo de comando incorreto:

ADD STN: TYPE = DP, CAT#= 1 , STN#=8889,PEN # = 700D, COS#=0;

30 - USO DE VALOR DE PARÂMETRO COMO COMANDO

- Descrição:

Substituição indevida de um comando por um valor de parâmetro existente na linguagem, devido à semelhança de nomes ou outra razão qualquer.

- Exemplo de comando incorreto:

DTMF STNASSN: TYPE = DP, CAT#= 1 , STN#=8889,PEN # = 700D, COS#=0;

31 - USO DE VALOR DE PARÂMETRO COMO PARÂMETRO

- Descrição:

Substituição indevida de um parâmetro por um valor de parâmetro existente na linguagem, devido à semelhança de nomes ou outra razão qualquer.

- Exemplo de comando incorreto:

ADD STNASSN: DTMF = DP, CAT#= 1 , STN#=8889,PEN # = 700D, COS#=0;

32 - USO DE VALOR DE PARÂMETRO COMO OBJETO

- Descrição:

Substituição indevida de um objeto por um valor de parâmetro existente na linguagem, devido à semelhança de nomes ou outra razão qualquer.

- Exemplo de comando incorreto:

ADD DTMF: TYPE = DP, CAT#= 1 , STN#=8889,PEN # = 700D, COS#=0;

33 - INVERSÃO (2 A 2) POSSÍVEL DE VALORES DE PARÂMETROS (VALP1 SUBCONJ VALP2) ??????

- Descrição:

Erro por inversão de valores de parâmetros, onde um deles e exclusivo (2 a 2) DE UM dos parâmetros envolvidos (P1), e foi atribuído ao outro parâmetro (P2). O valor atribuído a P1, pode ser valor-possível (no futuro-->"típico") tanto de P1 quanto de P2.

- Exemplo de comando incorreto:

ADD STNASSN: STN#=**700D**, TYPE = DP, CAT#= 1 , PEN # = **8889**, COS#=0;

|	|
P2	P1

(Quando "poderia" ser, por exemplo:

ADD STNASSN: STN#=**8889**, TYPE = DP, CAT#= 1 , PEN # = **700D**, COS#=0;)

- Rotina de correção:

Corrige_Erro_Inversao_Possiv_Val_Param(...) -> Ocorreu_Erro_ -> Msg_Erro_

ANEXO 3

LOGS DE SESSÕES DE TREINAMENTO

```
+--------------------------------------------------------------+
|                                                              |
| EQUITEL S/A.                                                 |
|                                                              |
| O.M.T. (Operation and Maintenance Terminal) - SPX 2000       |
|                                                              |
| BEGINNING DATE: 10/05/1996 TIME: 09:26:26                    |
|                                                              |
| FILE NAME: C:\OMTOUT\SPX_A\HISTORY.O08                       |
|                                                              |
| NUMBER  : 08                                                 |
|                                                              |
+--------------------------------------------------------------+

AC-CC0-SPX2000 V1501 PL:00.12 CUSTOMER: EQUITEL SITE: CURITIBA PR
PLEASE ENTER PASSWORD> GROUP 11 DEPENDABI1Y    ERRORS1    1 -
ERROR_194ExJ
AC-CC0-SPX2000 V1501 PL:00.12 CUSTOMER: EQUITEL SITE: CURITIBA PR
PLEASE ENTER PASSWORD>
 PEF SESSION TERMINATED AT 22-02-02 34:49:22
AC-CC0-SPX2000 V1501 PL:00.12 CUSTOMER: EQUITEL SITE: CURITIBA PR
PLEASE ENTER PASSWORD>
 PEF SESSION TERMINATED AT 22-02-02 34:49:24
AC-CC0-SPX2000 V1501 PL:00.12 CUSTOMER: EQUITEL SITE: CURITIBA PR
PLEASE ENTER PASSWORD>
22-02-02 34:49:28
>DIS DEVDATA

PORT EQUIPMENT NUMBER (WXYZ)=
LST H(HEX) OR T(DEC)=H

 PEF SESSION TERMINATED AT 22-02-02 34:50:45
AC-CC0-SPX2000 V1501 PL:00.12 CUSTOMER: EQUITEL SITE: CURITIBA PR
PLEASE ENTER PASSWORD>
22-02-02 34:50:49
>DIS STNASSN;
 DISPLAY STNASSN 22-02-02 34:51:10

                    RST  RST  WRM HOT FEAT      CHARGE
STN  PEN TYPE  COS  CAT  ORIG GRP    CAT   LNE LNE  ATTR  LST# METER
1155 7020 DTMF  0   1   ---- ---   0000   --- ---  000000 0020  00000213
1266 7002 DTMF  0   1   ---- ---   0000   --- ---  000000 000A  00000017
1267 7004 DTMF  0   1   ---- ---   0000   --- ---  000000 000C  00000021
```

```
1268 700A DTMF  0   1  ---- ---   0000   --- ---  000000 0012  00000013
1269 700C DTMF  0   1  ---- ---   0000   --- ---  000000 0014  00000002
1270 700F DTMF  0   1  ---- ---   0000   --- ---  000000 0017  00000044
2155 7022 DTMF  0   5  ---- ---   0000   --- ---  000000 0022  00000109
3155 7024 DTMF  0   5  ---- ---   0000   --- ---  000000 0024  00000099
4155 702A DTMF  0   3  ---- ---   0000   --- ---  000000 002A  00000015
5155 702C DTMF  0   3  ---- ---   0000   --- ---  000000 002C  00000000
6155 702F  DP   0   4  ---- ---   0000   --- ---  000000 002F  00000000

DISPLAY COMPLETE
>DIS STNASSN;
 DISPLAY STNASSN 22-02-02 35:01:06

                       RST   RST   WRM HOT  FEAT       CHARGE
 STN  PEN TYPE  COS  CAT  ORIG  GRP    CAT   LNE LNE  ATTR  LST#  METER
1155 7020 DTMF  0   1  ---- ---   0000   --- ---  000000 0020  00000213
1266 7002 DTMF  0   1  ---- ---   0000   --- ---  000000 000A  00000017
1267 7004 DTMF  0   1  ---- ---   0000   --- ---  000000 000C  00000021
1268 700A DTMF  0   1  ---- ---   0000   --- ---  000000 0012  00000013
1269 700C DTMF  0   1  ---- ---   0000   --- ---  000000 0014  00000002
1270 700F DTMF  0   1  ---- ---   0000   --- ---  000000 0017  00000044
2155 7022 DTMF  0   5  ---- ---   0000   --- ---  000000 0022  00000109
3155 7024 DTMF  0   5  ---- ---   0000   --- ---  000000 0024  00000099
4155 702A DTMF  0   3  ---- ---   0000   --- ---  000000 002A  00000015
5155 702C DTMF  0   3  ---- ---   0000   --- ---  000000 002C  00000000
6155 702F  DP   0   4  ---- ---   0000   --- ---  000000 002F  00000000

DISPLAY COMPLETE
>DIS SERVICE;
 DISPLAY SERVICE 22-02-02 35:24:00

EITHER PEN-, LST- OR EXTN NUMBER MUST BE ENTERED
PORT EQUIPMENT NUMBER (WXYZ)=7000 702F
 DISPLAY SERVICE 22-02-02 35:24:17

PEN 7002  IN SERVICE
PEN 7004  IN SERVICE
PEN 700A  IN SERVICE
PEN 700C  IN SERVICE
PEN 700F  IN SERVICE
PEN 7020  IN SERVICE
PEN 7022  IN SERVICE
PEN 7024  IN SERVICE
PEN 702A  IN SERVICE
PEN 702C  OUT OF SERVICE CRAFT
PEN 702F  OUT OF SERVICE CRAFT
END OF SERVICE DISPLAY
>DIS STNASSN;
 DISPLAY STNASSN 22-02-02 35:24:40

                       RST   RST   WRM HOT  FEAT       CHARGE
 STN  PEN TYPE  COS  CAT  ORIG  GRP    CAT   LNE LNE  ATTR  LST#  METER
1155 7020 DTMF  0   1  ---- ---   0000   --- ---  000000 0020  00000213
1266 7002 DTMF  0   1  ---- ---   0000   --- ---  000000 000A  00000017
1267 7004 DTMF  0   1  ---- ---   0000   --- ---  000000 000C  00000021
```

```
1268 700A DTMF  0   1  ---- ---   0000   --- ---  000000 0012  00000013
1269 700C DTMF  0   1  ---- ---   0000   --- ---  000000 0014  00000002
1270 700F DTMF  0   1  ---- ---   0000   --- ---  000000 0017  00000044
2155 7022 DTMF  0   5  ---- ---   0000   --- ---  000000 0022  00000109
3155 7024 DTMF  0   5  ---- ---   0000   --- ---  000000 0024  00000099
4155 702A DTMF  0   3  ---- ---   0000   --- ---  000000 002A  00000015
5155 702C DTMF  0   3  ---- ---   0000   --- ---  000000 002C  00000000
6155 702F  DP   0   4  ---- ---   0000   --- ---  000000 002F  00000000

DISPLAY COMPLETE
>CHA STNASSN:OSTN=1155,STN#=1000;
 CHANGE  STNASSN 22-02-02 35:25:49

CHARGE METER = 00000213
CHGE STN 1000  COMPLETE

>DIS STNASSN;
 DISPLAY STNASSN 22-02-02 35:26:05

                          RST    RST   WRM HOT  FEAT        CHARGE
 STN  PEN TYPE  COS  CAT  ORIG  GRP     CAT   LNE LNE  ATTR   LST#  METER
1000 7020 DTMF  0   1  ---- ---   0000   --- ---  000000 0020  00000213
1266 7002 DTMF  0   1  ---- ---   0000   --- ---  000000 000A  00000017
1267 7004 DTMF  0   1  ---- ---   0000   --- ---  000000 000C  00000021
1268 700A DTMF  0   1  ---- ---   0000   --- ---  000000 0012  00000013
1269 700C DTMF  0   1  ---- ---   0000   --- ---  000000 0014  00000002
1270 700F DTMF  0   1  ---- ---   0000   --- ---  000000 0017  00000044
2155 7022 DTMF  0   5  ---- ---   0000   --- ---  000000 0022  00000109
3155 7024 DTMF  0   5  ---- ---   0000   --- ---  000000 0024  00000099
4155 702A DTMF  0   3  ---- ---   0000   --- ---  000000 002A  00000015
5155 702C DTMF  0   3  ---- ---   0000   --- ---  000000 002C  00000000
6155 702F  DP   0   4  ---- ---   0000   --- ---  000000 002F  00000000

DISPLAY COMPLETE

.........

>CHA CUSTNAME

 PEF ERROR # 5
INVALID / UNKNOWN COMMAND NAME

>DIS DTMFASSN;
 DISPLAY DTMFASSN 22-02-03 44:01:32

ASSIGNED DTMF RECEIVERS BY PEN
7040 7042 7044 7046
END OF DISPLAY
>DIS SERVICE 7040 7046
PORT EQUIPMENT NUMBER (WXYZ)=7040 7046
MODE (<CR> CURRENT SERVICE MODE)=;
 DISPLAY SERVICE 22-02-03 44:02:34

PEN 7040  IN SERVICE
PEN 7042  IN SERVICE
```

```
PEN 7044  IN SERVICE
PEN 7046  IN SERVICE
END OF SERVICE DISPLAY
>DIS STNASSN;
 DISPLAY STNASSN 22-02-03 44:03:52

                         RST    RST    WRM HOT  FEAT         CHARGE
 STN  PEN TYPE  COS  CAT  ORIG  GRP    CAT   LNE LNE  ATTR   LST#  METER
1000 7020 DTMF  0    1  ---- ---   0000   --- --- 000000 0020  00000213
1002 7022 DTMF  0    5  ---- ---   0000   --- --- 000000 0022  00000109
1004 7024 DTMF  0    5  ---- ---   0000   --- --- 000000 0024  00000099
1010 702A DTMF  0    3  ---- ---   0000   --- --- 000000 002A  00000015
1012 702C DTMF  0    3  ---- ---   0000   --- --- 000000 002C  00000000
1015 702F  DP   0    4  ---- ---   0000   --- --- 000000 002F  00000000

DISPLAY COMPLETE
>DIS SERVICE
PORT EQUIPMENT NUMBER (WXYZ)=7000 7040
MODE (<CR> CURRENT SERVICE MODE)=;
 DISPLAY SERVICE 22-02-03 44:09:49

PEN 7020  IN SERVICE
PEN 7022  IN SERVICE
PEN 7024  IN SERVICE
PEN 702A  IN SERVICE
PEN 702C  IN SERVICE
PEN 702F  IN SERVICE
PEN 7040  IN SERVICE
END OF SERVICE DISPLAY
>CHA SERVICE:MODE=N,TYPE=X,EXTN=7020 702F

 PEF ERROR # 9
EXCESS PARAMETER VALUES
EXTENSION NUMBER?=7020;
 CHANGE  SERVICE 22-02-03 44:11:13

ERROR: ENTERED NUMBER NOT A STATION NUMBER.
EXTENSION NUMBER?=1000 1012

 PEF ERROR # 9
EXCESS PARAMETER VALUES

>CHA SERVICE:MODE=N,TYPE=X,EXTN=1000;
 CHANGE  SERVICE 22-02-03 44:11:48


SERVICES CHANGES ARE COMPLETED
>CHA SERVICE:MODE=N,TYPE=X,EXTN=1002;
 CHANGE  SERVICE 22-02-03 44:11:57


SERVICES CHANGES ARE COMPLETED
>CHA SERVICE:MODE=N,TYPE=X,EXTN=1004;
 CHANGE  SERVICE 22-02-03 44:12:13
```

```
SERVICES CHANGES ARE COMPLETED
>CHA SERVICE:MODE=N,TYPE=X,EXTN=1010;
 CHANGE  SERVICE 22-02-03 44:12:23


SERVICES CHANGES ARE COMPLETED
>

>CHA SERVICE:MODE=N,TYPE=X,EXTN=1012;
 CHANGE  SERVICE 22-02-03 44:12:36


SERVICES CHANGES ARE COMPLETED
>DEL STNASSN
STATION EXTENSION NUMBER=1000
CHANGE TO OTHER BU ? (Y)=;
 DELETE  STNASSN 22-02-03 44:13:14

CHARGE METER = 00000213
DEL STN COMPLETE

>DEL STNASSN:STN#=1002;
 DELETE  STNASSN 22-02-03 44:13:28

CHARGE METER = 00000109
DEL STN COMPLETE

>DEL STNASSN:STN#=1002;
 DELETE  STNASSN 22-02-03 44:13:45

NUMBER NOT A STATION
STATION EXTENSION NUMBER=

 PEF SESSION TERMINATED AT 22-02-03 44:13:48
AC-CC0-SPX2000 V1501 PL:00.12 CUSTOMER: EQUITEL  SITE: CURITIBA  PR
PLEASE ENTER PASSWORD>
22-02-03 44:13:52
>DIS SERVICE:MODE=I,TYPE=P,PEN=7000;

 PEF ERROR # 8
INVALID PARAMETER VALUE
DI DISPLAY SERVICE 22-02-03 44:14:53
S SERVICE:MODE=I,T
EITHER PEN-, LST- OR EXTN NUMBER MUST BE ENTEREDYPE=P,PEN=7002;

PORT EQUIPMENT NUMBER (WXYZ)=
 PEF ERROR # 8
INVALID PARAMETER VALUE

>DIS SERVICE:MODE=I,TYPE=P,PEN=7002;

 PEF ERROR # 8
INVALID PARAMETER VALUE
 DISPLAY SERVICE 22-02-03 44:15:02
```

```
EITHER PEN-, LST- OR EXTN NUMBER MUST BE ENTERED
PORT EQUIPMENT NUMBER (WXYZ)=

>DIS SERVICE:MODE=I,TYPE=P,PEN=7002;

 PEF ERROR # 8
INVALID PARAMETER VALUE
 DISPLAY SERVICE 22-02-03 44:15:27

EITHER PEN-, LST- OR EXTN NUMBER MUST BE ENTERED
PORT EQUIPMENT NUMBER (WXYZ)=

>DIS SERVICE:PEN=7000 700F
MODE (<CR> CURRENT SERVICE MODE)=;
 DISPLAY SERVICE 22-02-03 44:15:54

PORT EQUIPMENT NUMBER IS CURRENTLY UNASSIGNED
PORT EQUIPMENT NUMBER (WXYZ)=;
 DISPLAY SERVICE 22-02-03 44:16:02

EITHER PEN-, LST- OR EXTN NUMBER MUST BE ENTERED
PORT EQUIPMENT NUMBER (WXYZ)=7002
 DISPLAY SERVICE 22-02-03 44:16:06

PORT EQUIPMENT NUMBER IS CURRENTLY UNASSIGNED
PORT EQUIPMENT NUMBER (WXYZ)=

>DIS SERVICE
PORT EQUIPMENT NUMBER (WXYZ)=7000 702F
MODE (<CR> CURRENT SERVICE MODE)=;
 DISPLAY SERVICE 22-02-03 44:17:23

PEN 7024  OUT OF SERVICE CRAFT
PEN 702A  OUT OF SERVICE CRAFT
PEN 702C  OUT OF SERVICE CRAFT
PEN 702F  IN SERVICE
END OF SERVICE DISPLAY
>CHA SERVICE:MODE=N,TYPE=P,PEN=702F;
 CHANGE  SERVICE 22-02-03 44:18:04

SERVICES CHANGES ARE COMPLETED
>DEL STNASSN:STN#=1012;
 DELETE  STNASSN 22-02-03 44:22:58

CHARGE METER = 00000000
DEL STN COMPLETE

>DIS STNASSN;
 DISPLAY STNASSN 22-02-03 44:23:09

                     RST    RST   WRM HOT  FEAT       CHARGE
 STN  PEN TYPE  COS  CAT  ORIG  GRP    CAT   LNE LNE  ATTR  LST#  METER
1015 702F  DP    0    4   ---- ---   0000    --- ---  000000 002F  00000000
```

```
DISPLAY COMPLETE
>DEL STNASSN:STN#=1015
CHANGE TO OTHER BU ? (Y)=;
 DELETE  STNASSN 22-02-03 44:23:40

CHARGE METER = 00000000
DEL STN COMPLETE


>
```

SEGUNDO LOG

```
+--------------------------------------------------------------+
|                                                              |
| EQUITEL S/A.                                                 |
|                                                              |
| O.M.T. (Operation and Maintenance Terminal) - SPX 2000       |
|                                                              |
| BEGINNING DATE: 17/05/1996  TIME:  13:54:37                  |
|                                                              |
| FILE NAME: C:\OMTOUT\SPX_A\HISTORY.O15                       |
|                                                              |
| NUMBER   : 15                                                |
|                                                              |
+--------------------------------------------------------------+


AC-CC0-SPX2000 V1501 PL:00.12 CUSTOMER: EQUITEL  SITE: CURITIBA  PR
PLEASE ENTER PASSWORD>□
***************************** 17/05/1996 *** 13:54:40 ****************************
BU 0 ALARM GROUP 14 : DEPENDABILITY   ERRORS              - ERROR_ID :  1461
*************************************************************************************

 PEF SESSION TERMINATED AT 22-02-02 34:47:45
AC-CC0-SPX2000 V1501 PL:00.12 CUSTOMER: EQUITEL  SITE: CURITIBA  PR
PLEASE ENTER PASSWORD>
22-02-02 34:47:48
>DIS ALMDATA;
 DISPLAY ALMDATA 22-02-02 34:48:04

** FAILURE HISTORY MESSAGES **
SW TRAP    1476 ( 0) 4525 1965 4585 8500 09/13 13:21
FUSE/CONV. FAIL.(19)               02/02 02:23
DIU  0 ALM 1153 ( 9) PEN=7200 ,OFF       02/02 02:23
LTU CLOCK FAIL. (18) SHELF=05          02/02 02:24
AUDIT     1461 (18) 0000 0000 0000 0000 02/02 02:25
AUDIT     1461 (18) 0000 0000 0000 0000 02/02 02:27
AUDIT     1461 (18) 0000 0000 0000 0000 02/02 02:29
AUDIT     1461 (18) 0000 0000 0000 0000 02/02 02:31
AUDIT     1461 (18) 0000 0000 0000 0000 02/02 02:33
AUDIT     1461 (18) 0000 0000 0000 0000 02/02 02:35
AUDIT     1461 (18) 0000 0000 0000 0000 02/02 02:37
AUDIT     1461 (18) 0000 0000 0000 0000 02/02 02:39
```

```
AUDIT     1461 (18) 0000 0000 0000 0000 02/02 02:41
AUDIT     1461 (18) 0000 0000 0000 0000 02/02 02:43
AUDIT     1461 (18) 0000 0000 0000 0000 02/02 02:45
AUDIT     1461 (18) 0000 0000 0000 0000 02/02 02:47
**   END FAILURE HISTORY   **
>
........
>CHA STNASSN:OSTN=1000,TYPE=HOTLINE,EHID=1;
 CHANGE  STNASSN 22-02-02 35:35:54

HOTLINE CANNOT HAVE LAST NUMBER REDIAL AND SPEED DIALING
CLASS OF SERVICE NUMBER (0-31)=0
 CHANGE  STNASSN 22-02-02 35:36:10

CHARGE METER = 00000008
CHGE STN 1000  COMPLETE
>DIS STNASSN:STN#=1000;
 DISPLAY STNASSN 22-02-02 35:36:49

                        RST    RST    WRM HOT  FEAT        CHARGE
 STN  PEN TYPE  COS  CAT  ORIG  GRP    CAT   LNE LNE  ATTR   LST#  METER
1000 7000 HTLN   0    1   ---- ---    0000    ---  1   000000 0008  00000008

DISPLAY COMPLETE
>DIS STNASSN;
 DISPLAY STNASSN 22-02-02 35:37:43

                        RST    RST    WRM HOT  FEAT        CHARGE
 STN  PEN TYPE  COS  CAT  ORIG  GRP    CAT   LNE LNE  ATTR   LST#  METER
1000 7000 HTLN   0    1   ---- ---    0000    ---  1   000000 0008  00000008
1002 7002 DTMF   0    1   ---- ---    0000    --- ---  000000 000A  00000017
1004 7004 DTMF   0    1   ---- ---    0000    --- ---  000000 000C  00000022
1010 700A  DP    0    1   ---- ---    0000    --- ---  000000 0012  00000013
1015 7007 DTMF   0    1    0   ---    0000    --- ---  000000 000F  00000002
1269 700C DTMF   0    1   ---- ---    0000    --- ---  000000 0014  00000002
1270 700F DTMF   0    1   ---- ---    0000    --- ---  000000 0017  00000044

DISPLAY COMPLETE
>DIS HOTLINE
HOTLINE DELAYED/IMMED? (D,I)=I
HOTLINE INDEX NUMBER (0-63)=1
 DISPLAY HOTLINE 22-02-02 35:40:10

 INDEX  DESTINATION DIGITS
   1    1015

 END NONDIAL DESTINATIONS
.........................
>DEL STNASSN
STATION EXTENSION NUMBER=1000
CHANGE TO OTHER BU ? (Y)=;
 DELETE  STNASSN 22-02-03 44:18:57

CHARGE METER = 00000009
DEL STN COMPLETE
```

```
>ADD STNASSN:STN#=1000,PEN#=7000,CAT#=1,COS#=0;
TYPE ? ( DP,DTMF,HOTLINE)=DP;
 ADD    STNASSN 22-02-03 44:20:34

CHARGE METER = 00000009
ADD STN  1000  COMPLETE
>CHA SRVCE

 PEF ERROR # 5
INVALID / UNKNOWN COMMAND NAME

>CHA SERV

 PEF ERROR # 5
INVALID / UNKNOWN COMMAND NAME

>CHA SERVICE:MODE=I,TYPE=P,PEN=7000;
 CHANGE  SERVICE 22-02-03 44:21:33


SERVICES CHANGES ARE COMPLETED
>CHA STNASSN:OSTN=1000,COS#=1;
 CHANGE  STNASSN 22-02-03 44:23:20

CHARGE METER = 00000009
CHGE STN 1000  COMPLETE
..................
>DIS STNASSN:OSTN=1002,COS#=3;

 PEF ERROR # 6
INVALID / UNKNOWN KEYWORD
 DISPLAY STNASSN 22-02-03 45:04:36

                     RST    RST   WRM HOT  FEAT        CHARGE
 STN  PEN TYPE  COS  CAT  ORIG  GRP    CAT   LNE LNE  ATTR   LST#  METER
1000 7000  DP    2    1   ---- ---    0000    --- ---  000000 0008  00000009
1002 7002 DTMF   2    1   ---- ---    0000    --- ---  000000 000A  00000017
1004 7004 DTMF   0    1   ---- ---    0000    --- ---  000000 000C  00000023
1010 700A  DP    0    1   ---- ---    0000    --- ---  000000 0012  00000013
1015 7007 DTMF   0    1    0   ---    0000    --- ---  000000 000F  00000003
1269 700C DTMF   0    1   ---- ---    0000    --- ---  000000 0014  00000002
1270 700F DTMF   0    1   ---- ---    0000    --- ---  000000 0017  00000044

DISPLAY COMPLETE
>DIS FEATACC:TYPE=F,FEAT=XFERAUTO;

 PEF ERROR # 8
INVALID PARAMETER VALUE
FEATURE TO BE DISPLAYED=
>CHA STNASSN:OSTN=1002,COS

 PEF ERROR # 6
INVALID / UNKNOWN KEYWORD
```

NEW STN EXTN NUM (<CR>=NO CHGE)=

>CHA STNASSN:OSTN=1002,COS#=0
NEW STN EXTN NUM (<CR>=NO CHGE)=;
CHANGE STNASSN 22-02-03 45:34:48

CHARGE METER = 00000022
CHGE STN 1002 COMPLETE
>CHA STNASSN:OSTN=10042,COS#=0;

PEF ERROR # 8
INVALID PARAMETER VALUE
OLD STATION EXTENSION NUMBER=1004
CHANGE STNASSN 22-02-03 45:35:03

CHARGE METER = 00000023
CHGE STN 1004 COMPLETE
>CHA STNASSN:OSTN=1015,COS#=0;
CHANGE STNASSN 22-02-03 45:35:13

CHARGE METER = 00000006
CHGE STN 1015 COMPLETE
>DIS STNASSN;
DISPLAY STNASSN 22-02-02 34:52:04

```
                      RST   RST   WRM HOT FEAT        CHARGE
STN  PEN TYPE  COS  CAT  ORIG  GRP    CAT   LNE LNE  ATTR  LST#  METER
1000 7000  DP    2    1  ---- ---   0000   --- ---  000000 0008  00000009
1002 7002 DTMF   3    1  ---- ---   0000   --- ---  000000 000A  00000022
1004 7004 DTMF   3    1  ---- ---   0000   --- ---  000000 000C  00000026
1010 700A  DP    0    1  ---- ---   0000   --- ---  000000 0012  00000013
1015 7007 DTMF   0    1   0   ---   0000   --- ---  000000 000F  00000015
1269 700C DTMF   0    1  ---- ---   0000   --- ---  000000 0014  00000002
1270 700F DTMF   0    1  ---- ---   0000   --- ---  000000 0017  00000044
```

DISPLAY COMPLETE
>DIS SERICE;

PEF ERROR # 5
INVALID / UNKNOWN COMMAND NAME

>DIS SERVICE:PEN#=7000 701F;

PEF ERROR # 6
INVALID / UNKNOWN KEYWORD
DISPLAY SERVICE 22-02-02 34:54:17

EITHER PEN-, LST- OR EXTN NUMBER MUST BE ENTERED
PORT EQUIPMENT NUMBER (WXYZ)=7000 701F;
DISPLAY SERVICE 22-02-02 34:54:29

PEN 7000 OUT OF SERVICE AUTO
PEN 7002 OUT OF SERVICE AUTO
PEN 7004 OUT OF SERVICE AUTO
PEN 7007 OUT OF SERVICE AUTO

```
PEN 700A  OUT OF SERVICE AUTO
PEN 700C  OUT OF SERVICE AUTO
PEN 700F  OUT OF SERVICE AUTO
END OF SERVICE DISPLAY
>DIS SERVICE:PEN=7000 700F;

 PEF SESSION TERMINATED AT 22-02-02 34:55:21
AC-CC0-SPX2000 V1501 PL:00.12 CUSTOMER: EQUITEL  SITE: CURITIBA  PR
PLEASE ENTER PASSWORD>
22-02-02 34:55:27
>DIS SERVICE:PEN=7000 700F;
 DISPLAY SERVICE 22-02-02 34:55:30

PEN 7000  OUT OF SERVICE AUTO
PEN 7002  OUT OF SERVICE AUTO
PEN 7004  OUT OF SERVICE AUTO
PEN 7007  OUT OF SERVICE AUTO
PEN 700A  OUT OF SERVICE AUTO
PEN 700C  OUT OF SERVICE AUTO
PEN 700F  OUT OF SERVICE AUTO
END OF SERVICE DISPLAY
>DEL STNASSN:STN=1000;

 PEF ERROR # 6
INVALID / UNKNOWN KEYWORD
STATION EXTENSION NUMBER=1000;
 DELETE  STNASSN 22-02-02 34:56:01

STATION IN SERVICE
STATION EXTENSION NUMBER=1002;
 DELETE  STNASSN 22-02-02 34:56:07

STATION IN SERVICE
STATION EXTENSION NUMBER=1004;
 DELETE  STNASSN 22-02-02 34:56:11

STATION IN SERVICE
STATION EXTENSION NUMBER=

>
..............
>DIS STNASS;

 PEF ERROR # 5
INVALID / UNKNOWN COMMAND NAME

>DIS STNASSN;
 DISPLAY STNASSN 96-04-09 10:05:30

                        RST    RST    WRM HOT  FEAT       CHARGE
 STN  PEN TYPE  COS  CAT  ORIG  GRP    CAT   LNE LNE  ATTR   LST#  METER
1000 7000 DTMF   0    1   ---- ---    0000    --- ---  000000 0008  00000004
1002 7002 DTMF   0    1   ---- ---    0000    --- ---  000000 000A  00000002
1004 7004 DTMF   0    1   ---- ---    0000    --- ---  000000 000C  00000000
1005 7005 DTMF   0    1   ---- ---    0000    --- ---  000000 000D  00000000
```

```
1014 700E DTMF  0   1  ---- ---   0000   --- --- 000000 0016  00000000
1015 700F DTMF  0   1  ---- ---   0000   --- --- 000000 0017  00000000

DISPLAY COMPLETE
.........
>DIS SERVICE
PORT EQUIPMENT NUMBER (WXYZ)=7002
MODE (<CR> CURRENT SERVICE MODE)=I

 PEF ERROR # 8
INVALID PARAMETER VALUE
MODE (<CR> CURRENT SERVICE MODE)=

>
.................
>DIS SNTASSN;

 PEF ERROR # 5
INVALID / UNKNOWN COMMAND NAME

>DIS STNASSN;
 DISPLAY STNASSN 96-04-10 13:23:17

                     RST   RST   WRM HOT  FEAT        CHARGE
 STN  PEN TYPE  COS  CAT  ORIG  GRP    CAT   LNE LNE  ATTR  LST#  METER
1000 7000 DTMF  0   1  ---- ---   0000   --- --- 000000 0008  00000011
1002 7002 DTMF  0   1  ---- ---   0000   --- --- 000000 000A  00000006
1004 7004 DTMF  0   1  ---- ---   0000   --- --- 000000 000C  00000005
1005 7005 DTMF  0   1  ---- ---   0000   --- --- 000000 000D  00000000
1014 700E DTMF  0   1  ---- ---   0000   --- --- 000000 0016  00000000
1015 700F DTMF  0   1  ---- ---   0000   --- --- 000000 0017  00000000

DISPLAY COMPLETE
................
>DIS SERVICE:PEN=7108;
 DISPLAY SERVICE 96-04-10 13:25:27

PORT EQUIPMENT NUMBER IS CURRENTLY UNASSIGNED
PORT EQUIPMENT NUMBER (WXYZ)=DIS SERVICE:PEN=7100;

 PEF ERROR # 8
INVALID PARAMETER VALUE

>DIS SERVICE:PEN=7100;
 DISPLAY SERVICE 96-04-10 13:26:01

PEN 7100  IN SERVICE
END OF SERVICE DISPLAY
>

>CHA SERVICE:MODE=N,TYPE=P,PEN#=7102 7106;

 PEF ERROR # 6
INVALID / UNKNOWN KEYWORD
 CHANGE  SERVICE 96-04-10 13:28:58
```

EITHER PEN-, LST- OR EXTN NUMBER MUST BE ENTERED
PORT EQUIPMENT NUMBER (WXYZ)=

>CHA SERVICE:MODE=N,TYPE=P,PEN=7102 7106;
CHANGE SERVICE 96-04-10 13:29:21

SERVICES CHANGES ARE COMPLETED
....

REFERÊNCIAS BIBLIOGRÁFICAS

ADAM, A.; LAWRENT, J.P. **LAURA:** A system to debug student programs. Journal of Artificial Intelligence, v.15, p. 75-122, 1980

ANDERSON, J.R. **Cognitive psychology and intelligent tutoring.** In: **Proceedings of the Cognitive Science Society Conference.** Boulder, Colorado. p.37-43, 1984

ANDERSON, J.R.; BOYLE, F.; CORBETT, A.I.; et al. **Cognitive Modeling and Intelligent Tutoring.** Artificial Intelligence 42, 1980

ANDERSON, J.R.; BOYLE, C.F.; REISER, B.J. **Intelligent Tutoring Systems.** Science, v. 228, n. 4698. p. 456-462, 1985

BARR, A.; BEARD, M.; ATKINSON, R.C. **The computer as a Tutorial Laboratory:** the Stanford BIP project. In: **International Journal of Man-Machine Studies** v. 8, p. 567-596, 1976

BARRET, A.; RAMSAY, A.; SLOMAN, A. **POP-11 a Practical Language for Artificial Intelligence.** Ellis Horwood, 1985

BLESSING, S.B. **A Programming by Demonstration Authoring Tool for Model-Tracing Tutors.** International Journal of Artificial Intelligence in Education, v. 8, p. 233-261, 1997

BONAR, Jeffrey R. **Bite-sized intelligent tutoring.** LRDC Working document, 1985

BUCHANAN, B.G.; SHORTLIFFE, E.J. **Rule-Based Expert Systems:** The MYCIN Experiments of the Stanford Heuristic Programming Project. Reading, MA: Addison-Wesley, 1984

du BOULAY, Benedict. **Intelligent Systems for Teaching Programming Artificial Intelligence Tools in Education**. 1988

du BOULAY, Benedict; SOTHCOTT, Christopher. **Computers Teaching Programming:** an introductory survey of the field. In: LAWLER, R.W.; YASDANI, M. **Artificial Intelligence and Education:** learning environment and tutoring systems, v. 1, cap. 16, p. 345-372, 1987

CLANCEY, W.J. **GUIDON**. In: **Journal of Computer-Based Instruction**. v. 10, n.1, p. 8-14, 1983

CLANCEY, W.J. **Knowledge-based Tutoring:** The GUIDON Program. Cambridge, Massachusetts: MIT Press, 1987

DIRENE, Alexandre I. **Methodology and Tools for Designing Concept Tutoring Systems**. In: **The Third White House Papers.** CSRP 172, School of Cognitive and Computing Sciences - University of Sussex, 1990

DIRENE, Alexandre I. **Intelligent Training Shells for the Operation of Digital Telephony Stations.** In: **Proceedings of AIED**, 1997

DIRENE, Alexandre, I.; NASCIMENTO, E.; PRETO, Tania M.; et al. **Sistemas Tutoriais para Assistir o Treinamento da Operação de Centrais de Comutação.** In: **Anais do Simpósio Brasileiro de Informática na Educação.** p. 167-149, 1997

GOLDSTEIN, I.P. **Summary of MYCROFT:** a system for understanding simple picture programs. In: **Artificial Intelligence**, v. 6, n. 3, p. 249-288, 1975

JOHNSON, W.L. **Intention-Based Diagnosis of Novice Programming Errors** - Research Notes in Artificial Intelligence. Morgan Kaufmann, 1986

MAJOR, N.; AINSWORTH, S; WOOD, D. **REDEEM:** Exploiting Symbiosis Between Psychology and Authoring Environments. International Journal of Artificial Intelligence in Education, v. 8, p. 317-340, 1997

MAJOR, N.; REICHGELT, H. **Using COCA to build an intelligent tutoring system in simple algebra.** Intelligent Tutoring Media, v. 2, n. 314, p. 159-169, 1991

MURRAY, T.; WOOLF, B.P. **A knowledge Acquisition Tool for Intelligent Computer Tutors**. SIGART Bulletin, v. 2, n. 2, 1990

MURRAY, Tom. **Authoring Intelligent Tutoring Systems:** An analysis of the state of the art. International Journal of Artificial Intelligence in Education, v. 10, p. 98-129, 1999

NICHOLSON, R.I.; SCOTT, P.J. **Computers and Education:** the software production problem. British Journal of Educational technology, v. 17, n. 1, p. 26-35, jan. 1986

NUNES, M.G.V.; HASEGAWA, R.; VIEIRA, F.M.C.; et al. **SASHE:** Sistema de Autoria e Suporte Hipermídia para Ensino. Notas do ICMSC, n. 33, USP, 1997

OUTI, Elaine C. **Uma aplicação independente de domínio para a simulação de parte das funções de uma central digital de telefonia.** Curitiba, 1997. Trabalho de Graduação (Bacharelado em Informática) - Departamento de Informática, Universidade Federal do Paraná

PAPERT, S. **Mindstorms:** Children, Computers and Powerful Ideas. New York: Basic Books, 1980

RAMADHAN, Haider; du BOULAY, Benedict. **Programming Environment for Novices**. In: LEMUT, Enrica. **Cognitive models and Intelligent Environments for Learning Programming**. Springer-Verlag, v. F111, p. 125-134, 1993

RICH, Elaine; KNIGHT, Kevin. **Inteligência Artificial.** 2. ed. Makron Books do Brasil Ed. LTDA, 1994

SHORTLIFFE, E.H. **Computer based medical consultations :** MYCIN. New York : Elsevier, 1976

SLEEMAN, D.H. **Asserting competence in basic algebra**. In: SLEEMAN, D.H. & BROWN, J.S.. **Intelligent Tutoring Systems.** London: Academic Press, 1982. p. 185-199

SLEEMAN, D. **Micro-SEARCH:** a "shell" for building systems to help students solve non-deterministic tasks. AI & instruction. Addison-Wesley, 1987a

SLEEMAN, D.H. **Pixie**: a shell for developing Intelligent Tutoring Systems. In: LAWLER, R. & YAZDANI, M.. **AI & education :** learning environments and ITS. Abblex Publishing, 1987b

SZABO, M.; POOHKAY, B. **Animation, Mathematics Achievement and Attitude Toward Computer Assisted Instruction:** an experiment. In: **Proceedings of ED-Media**, 1995

TEMPLETON, Brad **Alice Pascal**. www.templetons.com/brad/alice.html, 1986

WENGER, Etienne.; **Artificial Intelligence and Tutoring Systems.** Morgan Kaufmann Publishers Inc., 1987